Universitat de Lleida

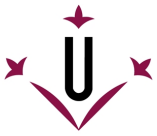
Grado en Ingeniería Informática

Trabajo fin de grado

Smart Agro Visualization Tool

Autor:
Javier Calvo García

Tutor:
Fernando Cores Prado



Universitat de Lleida

Índice

Índice de figuras

Índice de tablas

1. Introducción

Usualmente el sector agrícola es caracterizado como un mundo tradicional trabajado con una baja tecnificación. El auge de nuevas tecnologías y el anhelo de tener un control automatizado de los elementos que nos rodean ha permitido la monitorización y el análisis de parámetros ambientales en los campos de cultivo, para generar importantes mejoras en la productividad, optimizar los recursos del lugar y producir una sostenibilidad económica y medioambiental. En la actualidad, los equipos técnicos que crean un terreno inteligente están compuestos de dos tecnologías principales.
La primera de ellas son los sensores: la agricultura dispone de una variedad muy extensa de instrumentos agrarios que captan todo tipo de magnitudes físicas significativas para el cultivo. Sensores de temperatura, humedad, luminiscencia, viento y polución son los más populares de un gran abanico de medidores que, junto a una gestión informática, se centralizan y ejecutan de forma autónoma para una mejora potencial en el cultivo [5].

Figura 1: Estación de cálculo de humedad de la marca Dacom [1].

Otro factor importante en ésta tecnología es el crecimiento del llamado Internet de las Cosas[1] [6]. El concepto IoT contempla la conexión digital de objetos a través de Internet. Dichos objetos pueden ser meros elementos cotidianos (una nevera, unas zapatillas...) o dispositivos más complejos con funcionamiento propio como los sensores mencionados anteriormente. [7] Los recursos que proporciona la tecnología IoT en la conexión y interacción de los datos de sus elementos pueden ser muy útiles en el campo.

Los drones[2], la segunda tecnologia de los equipos agrícolas inteligentes, tienen un uso asegurado en el presente y futuro de la agricultura. Esta maquinaria

[1] (Internet of things, IoT), concepto de computación que contempla la interrelación entre máquinas mecánicas y digitales, objetos, animales y/o personas provistos de identificadores únicos a través de Internet.

[2] Vehículo aéreo no tripulado, (en inglés Unmanned Aerial Vehicle, UAV) es una aeronave que vuela sin tripulación propulsada por un motor de explosión ,eléctrico, o de reacción.

ofrece múltiples posibilidades para el sector agrícola: sobrevuelan cientos de hectáreas de forma precisa en poco tiempo y captan diversa información con los sensores que tienen incorporados.

Figura 2: Dron en acción en un campo de cultivo [2].

La información recogida sobre hidratación, temperatura, ritmo de crecimiento de los cultivos o localización prematura de enfermedades, entre otros tipos, es analizada posteriormente por software especializado en el ámbito para proporcionar un ahorro de costes significativo para los agricultores. Además, los drones pueden realizar funciones diferentes a la recolección de información, como arrojar productos químicos, controlar el riego o dispersar a los animales como espantapájaros improvisados.

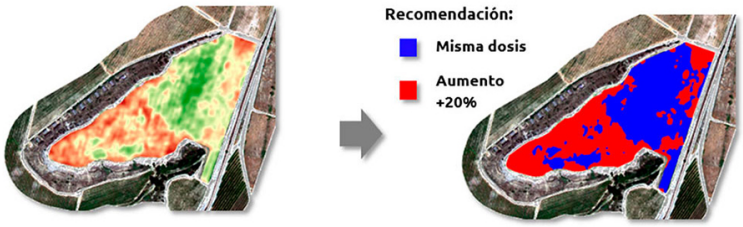


Figura 3: Fotografías de drones analizadas para una recomendación de abono [3].

Para poder interactuar con la información procesada de estos dos elementos, será necesario la utilización de alguna plataforma de datos estable que pueda controlar todos los datos registrados en el instante. La plataforma FIWARE ofrece un conjunto de servicios potentes para el desarrollo de aplicaciones inteligentes [8]. FIWARE consta de los llamados FIWARE Lab, entornos de creación no comerciales por internet para la innovación y experimentación de proyectos con tecnología FIWARE. FIWARE Lab se despliega sobre una red geográficamente distribuida de nodos federados aprovechando una amplia gama de infraestructuras experimentales, proporcionando recursos en la nube (sistemas operativos, instancias y sofware específico) a los usuarios que lo requieran. Con el uso de esta tecnología, los datos inteligentes del campo pueden estar disponibles en la nube de la instancia de FIWARE Lab, ser procesados allí mismo con recursos más potentes si se necesita, y podrán ser consultados en cualquier lugar a través de internet.

La presentación de los datos del campo es el último factor a tener en cuenta en la inteligencia de los campos de cultivo. El modo en que el cliente agricultor perciba la información de sensores y dornes es crucial para el incremento de beneficios de sus terrenos. La visualización de los datos tiene que ser entendible y simple para el usuario que la reciba, para que sepa en todo momento los cambios que deba realizar en el campo para mejorarlo. Una herramienta destacada en la exhibición de datos geográficos es la tecnología Liquid Galaxy[3]. Liquid Galaxy es un conjunto de ordenadores que ejecutan el programa Google Earth[4] para crear una experiencia inmersa sobre la geografia. Al programa Google Earth se le pueden importar ficheros KML[5] con datos personalizados para representar cualquier cosa que se desee [4].
En este proyecto se combinaran las tecnologías FIWARE y Liquid Galaxy, para desarrollar una aplicación que represente de forma automática los datos procesados recogidos por sensores y drones en los campos de cultivo. En otras palabras, sensores y drones enviarán la información que recolecten a una instancia personalizada de la plataforma FIWARE, y una aplicación servidor Django cogerá los datos de la instancia que se requiera el usuario a través de servirá una interfaz en navegador web que servirá a la vez la aplicación, para finalmente visualizar dichos datos en el Liquid Galaxy:

[3]Clúster de ordenadores con diversas pantallas que ejecutan el programa Google Earth de forma sincronizada para crear una experiencia inmersa [9]. Este sistema se usa como base para desarrollar proyectos de representación de datos, que consisten en la creación de aplicaciones que se ejecutan en un servidor que está conectado al ordenador central del clúster.

[4]Programa desarrollado por la empresa Google que muestra un globo virtual mapeado a través de la superposición de fotografias de satélites donde se muestra información geográfica.

[5]Keyhole Markup Language es un lenguaje basado en XML usado para la representación de datos geográficos. KML es la extensión que usa Google Earth para superponer figuras, imágenes y otras representaciónes en el mapa geográfico.

Figura 4: Liquid Galaxy en funcionamiento con cinco pantallas [4].

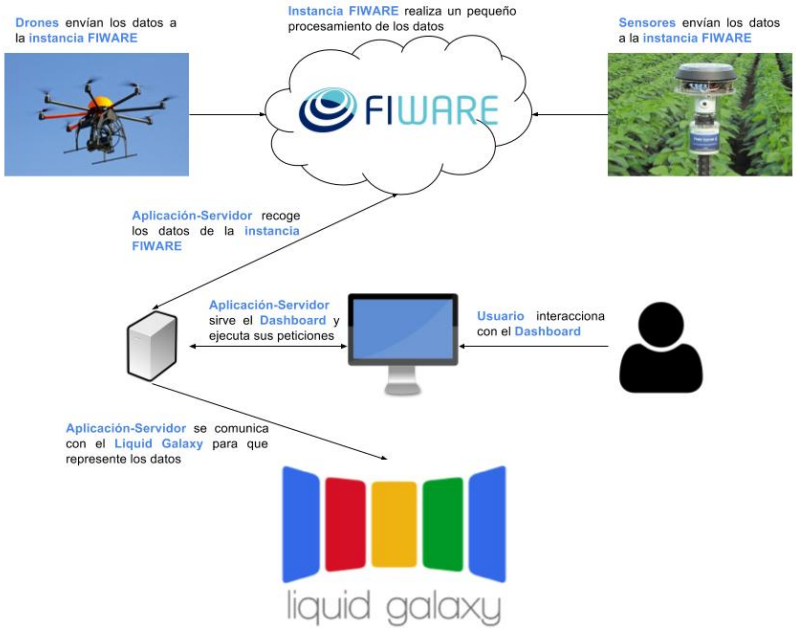


Figura 5: Estructura inicial de Smart Agro Visualization Tool.

1.1. Estudio previo

Todas las tecnologías presentadas anteriormente han requerido un estudio previo para ser utilizadas. Durante el transcurso del grado de Ingeniería Informática ya se ha trabajado con Python[6] continuamente, por lo que sólo se ha necesitado informarse sobre las librerías de éste lenguaje que podrían usarse en el desarrollo de la aplicación. Django[7], REST[8] y SSH[9] también se han visto en las diferentes asignaturas de la carrera, pero se ha precisado practicar con éstas tecnologías a fin de tener una pequeña soltura a la hora de programar con ellas.

Respecto a los diferentes tipos de datos[10] usados ya se habían usado antes, y no han requerido problemas para iniciar los primeros pasos del proyecto.

Por otro lado, toda tecnología relacionada con Liquid Galaxy era desconocida hasta la momento, por lo que ha sido necesario un aprendizaje desde cero. Para ello se instaló un sistema Liquid Galaxy de forma completa, incluyendo la instalación del sistema operativo en los ordenadores y la sincronización entre cada uno.

Por último, Google Earth ha sido otra tecnología que no se había trabajado con ella, aunque ésta sí que era conocida con anterioridad, exclusivamente como usuario de la aplicación. Para su estudio se ha practicado con la creación y importación de ficheros KML a Google Earth para visualizar pequeñas representaciones geográficas como puntos o animaciones de viajes por el mapa.

1.2. Objetivos

La realización del proyecto tiene como objetivos los siguientes puntos:

- Desarrollar una aplicación que muestre información útil al usuario en tiempo real.
- Implementar la aplicación para que se ejecute de la forma más fluida y rápida posible a las órdenes del usuario.
- Generar unas representaciones geográficas entendibles a cualquier usuario.

[6] Lenguaje de programación multiparadigma. Se utilizará este lenguaje para la programación de los distintos scripts (programas informáticos simples) que recojan y procesen los datos a representar.

[7] Framework de desarrollo web de código abierto escrito en lenguaje Python basado en el patrón de diseño Modelo-vista-controlador. Django será utilizado para implementar la aplicación servidor del proyecto que hospede el tratamiento de datos y su respectiva visualización.

[8] Transferencia de Estado Representacional (en inglés Representational State Transfer), estilo de arquitectura software utilizado por la mayoría de plataformas de datos abiertos.

[9] Protocolo de comunicación necesario para la comunicación Servidor – Liquid Galaxy que transmitirá los ficheros de representación de datos en el globo virtual.

[10] JSON, CSV y XML, entre otros, serán los tipos de archivos que se usaran en el proyecto para guardar la información que obtenga sensores y drones.

- Crear una respuesta (feedback) inteligible de forma continua a las acciones del usuario.
- Crear una experiencia interactiva única.

El proyecto también tiene estos objetivos personales:

- Aprendizaje des de cero de una nueva tecnología (*Liquid Galaxy*).
- Estudio en la recolecta y el procesamiento de datos abiertos.
- Práctica en la preparación, documentación e implementación de un proyecto tecnológico.

1.3. Tareas

A fin de cumplir los objetivos del proyecto, se ha dividido el desarrollo del proyecto en varias tareas a realizar, que se ejecutaran, en la medida de lo posible, en el orden descrito a continuación, y con la temporalización explicada más adelante:

- Preparación del proyecto:
  - Instalar entorno de trabajo, librerías y recursos necesarios.
  - Realizar estudio previo y práctica de las tecnologías relacionadas con el proyecto (Liquid Galaxy, Django y Fiware).
  - Planificar el desarrollo del proyecto (objetivos y temporización).
  - Analizar los casos de uso y requerimientos de la aplicación.
- Desarrollo del proyecto:
  - Implementar aplicación servidor Django:
    - Diseñar modelo de datos.
    - Estructurar direccionamiento de páginas.
    - Diseñar panel de datos en tiempo real (Dashboard).
    - Diseñar interfaz de usuario.
  - Programar scripts Python generadores de archivos KML para las representaciones geográficas.
  - Idear y implementar recogida de datos de la plataforma FIWARE.
  - Establecer comunicación entre servidor que aloje la aplicación y Liquid Galaxy.
- Presentación del proyecto:
  - Crear ejemplos de muestra codificados de casos de uso de la aplicación.
  - Documentar todo el proceso de desarrollo.

1.4. Planificación

Antes de idear la planificación del proyecto se ha de saber que ésta será variable durante todo el desarrollo del proyecto. Esto es debido a que puede surgir cambios en la implementación de cada tarea por la existencia de problemas. Además, la o las plataformas de datos de donde se recogerán los valores a representar pueden variar en el transcurso del proyecto, cambiando su estructura o su punto de recogida entre otros posibles cambios, por lo que éste factor será primordial para crear una planificación flexible que se pueda ver modificada creando el mínimo número de problemas en la implementación del trabajo.
La planificación está dividida en días, de lunes a viernes, a media jornada (horas) cada día. El plan temporal empieza en el mes de febrero, donde se confeccionará una preparación del proyecto con las siguientes fases: instalación del entorno de trabajo (1 semana), estudio y práctica previa de las tecnologías que se utilizaran (7-8 semanas), planificación de desarrollo (3-4 semanas) y análisis de casos de uso y requerimientos (2-3 semanas).
Una vez terminada la preparación aproximadamente a mediados de mayo, se iniciará el desarrollo del proyecto de la forma siguiente: la programación de los scripts Python y la recogida de datos de FIWARE tendrán la mayor duración del plan (12-14 semanas) ya que desarrollaran el núcleo del proyecto y deberán ejecutarse al mismo tiempo porque dependen entre sí (depende de los datos que se recojan, se crearan distintos tipos de representaciones y viceversa). A su vez, en el mes de junio empezará el desarrollo de la aplicación servidor Django con distintas sub-fases secuenciales (primero se diseñará el modelo de datos durante un mes, después se estructuraran las páginas a mostrar durante otro mes, y se acabará con el diseño del panel de datos, la interfaz y la conexión al Liquid Galaxy con otro mes aproximadamente).
Para acabar, las dos últimas semanas del plan se destinaran a finalizar la documentación y pulir la presentación del proyecto creando ejemplos de muestra codificados para mostrar sus posibles usos, y entregar el trabajo a inicios de setiembre.
Cabe comentar que la documentación del proyecto se realizará durante todo el desarrollo para poder registrar cualquier información importante que surja durante éste.

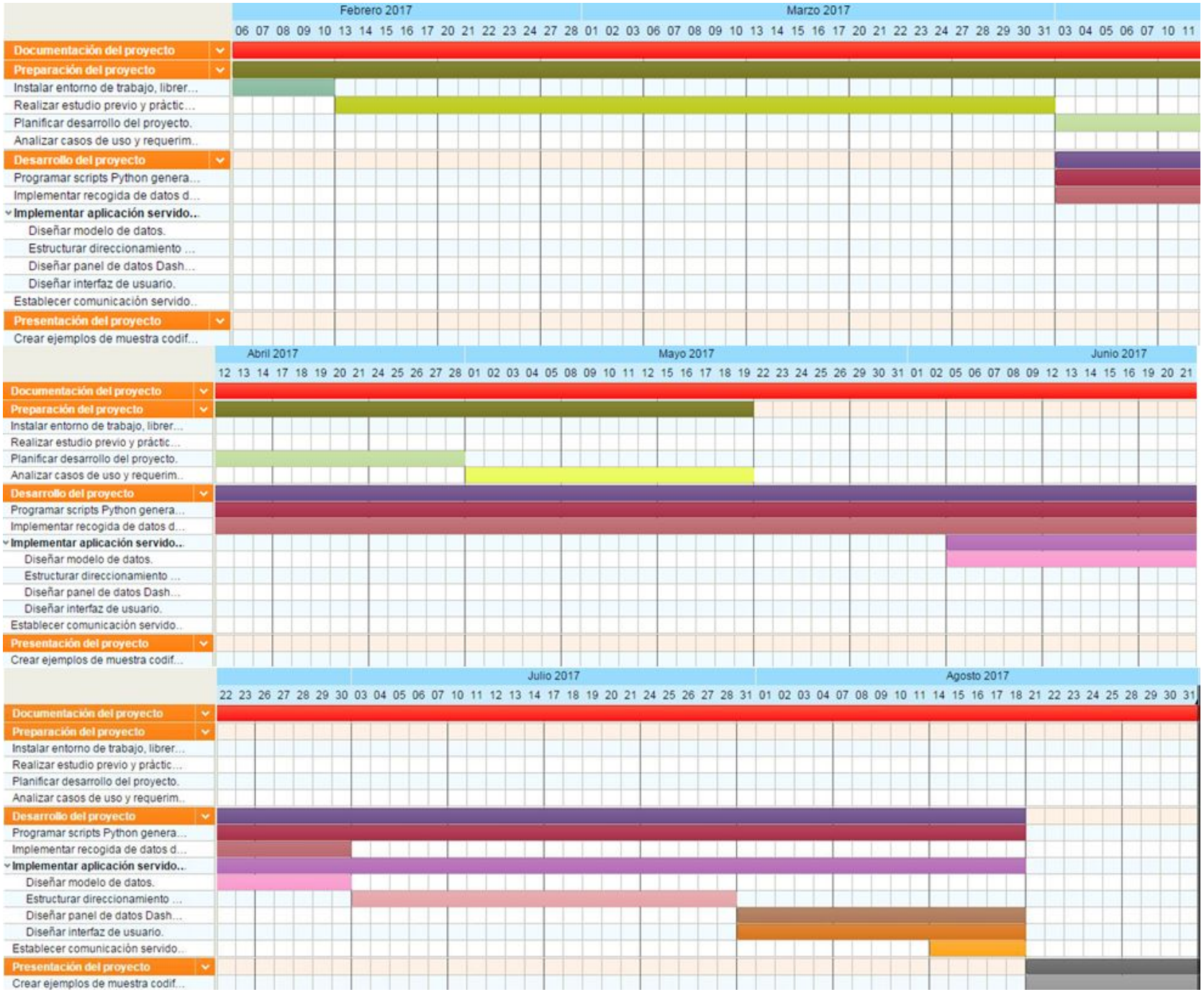


Figura 6: Planificación del desarrollo del proyecto.

1.5. Análisis de costes

El presupuesto de este proyecto se dividirá en dos campos a tener en cuenta: Coste de personal y Coste de software. No se ha tenido en cuenta el cálculo del coste de las posibles instalaciones o el hardware a utilizar, ya que éste está incluido en el gasto de personal.

1.5.1. Coste de personal

Lugar de trabajo	Número de horas	Coste por hora	Coste total (euros)
Programador	600	20	12000
Analista	100	40	4000
Redactor	50	30	1500

Tabla 1: Cálculo del coste de personal del proyecto.

1.5.2. Coste de software

No habrá coste alguno en el software de desarrollo del proyecto. Se desarrollará de forma abierta en el sistema operativo gratuito Ubuntu de Linux, usando el repositorio abierto GitHub y con programación de lenguajes abiertos y gratuitos. Se ha de comentar que el uso de la plataforma FIWARE no tendrá coste ya que se utilizarán recursos prestados de forma gratuita por la misma plataforma.

1.5.3. Coste final del proyecto

Tipo	Coste (euros)
Personal	17500
Software	0
Coste total	17500

Tabla 2: Cálculo del coste final del proyecto.

En el siguiente apartado se explicaran con detalle el origen, la estructura, funcionamiento y rol dentro del proyecto de cada tecnología usada en éste.

2. Estado del arte

2.1. Fuentes de información

Dado que no se consta de instrumentos de medida reales, los datos utilizados en el proyecto serán simulados, creando una demostración lo más aproximada a la realidad sobre estos datos. Para conseguir reproducir unos valores cercanos a la realidad, se ha realizado un pequeño estudio sobre el trabajo que ejecutan las empresas punteras del sector de la agricultura inteligente, donde generalmente la toma de datos se realiza con dos instrumentos técnicos: sensores y drones.

2.1.1. Sensores

Los sensores se distribuyen uniformemente sobre una plantación, donde miden y envían, en intervalos de tiempo muy cortos, los valores de las magnitudes físicas que se les ha configurado. Gracias a la publicación de empresas[11] de las características y funcionamiento de los sensores que comercializan se ha podido configurar la siguiente tabla de tipos de sensores, compuesta de los modelos más populares:

Magnitud	Medida	Rango de valores
Temperatura Aire	Grados Celsius (ºC)	0 - +65 ºC
Humedad Aire	Porcentaje (%)	0 - 100 %
Presión Aire	Pascales (Pa)	30 - 110 kPa
Temperatura Suelo	Grados Celsius (ºC)	-55 - 125 ºC
Humedad foliar	Porcentaje (%)	0 - 100 %
Radiación solar	$(\mu mol * m^{-2}s^{-1})$	$0 - 4000 \mu mol * m^{-2}s^{-1}$
Radiación ultravioleta	$(\mu mol * m^{-2}s^{-1})$	$0 - 300 \mu mol * m^{-2}s^{-1}$
Diámetro del tronco	Centímetros (cm)	Desde 2 cm
Diámetro del tallo	Centímetros (cm)	0 - 5 cm
Diámetro del fruto	Centímetros (cm)	0 - 11 cm
Anemómetro	Kilómetros/hora(km/h)	0 - 240km/h
Dirección del viento	Punto cardinal	16 puntos
Pluviómetro	Milímetros/hora(mm/h)	0 - 60 mm/h
Luminosidad	Lux (lx)	0.1 - 40000 lx
Ultrasonido	Centímetros (cm)	0 - 645 cm

Tabla 3: Tabla de magnitudes medidas por sensores de agricultura.

[11] La empresa más destacada en la búsqueda ha sido libelium [10], con su abanico de dispositivos Waspmote [11]

2.1.2. Drones

En el estudio de un campo de cultivo, los drones tienen la función de sobrevolar el terreno y realizar fotografías con planos cenitales[12]. Posteriormente, a través de programas especializados, realizan un procesamiento de estas fotografías para obtener imágenes analizadas de distinto tipo. Los análisis más comunes (que serán simulados a partir de fotografías originales de drones) entre los distintos tipos de plantaciones son los siguientes:

- Imagen Vegetación: Análisis del nivel de vegetación en el campo fotografiado. El análisis de vegetación provoca a la imagen un tono rojizo brillante, con diversas zonas más oscuras.
- Imagen Vigor: Análisis del nivel de fertilización del campo fotografiado. Visualmente éste análisis crea unos colores azulados, con zonas puntuales en otros tonos (amarillo y verde).
- Imagen Nitrógeno: Análisis de los niveles de nitrógeno del campo fotografiado. Éste tipo de imagen se caracteriza por un tono muy claro, con una subida muy alta del brillo de la copia.

2.2. Plataformas de visualización

Seguidamente se listan las tecnologías necesarias para visualizar los datos provenientes de las fuentes mencionas en el apartado anterior:

2.2.1. Liquid Galaxy

Liquid Galaxy es un proyecto de código abierto creado en 2008 por el empleado de Google Jason Holt. [13] Éste sistema combina pantallas de alta definición, múltiples ordenadores y el control en tres dimensiones Space Navigator para crear un experiencia inmersiva de navegación global. La visión del usuario, que se extiende a través de todas las pantallas, se curva para llenar la visión periférica de éste con el mundo, países o calles que esté navegando gracias al programa Google Earth. La navegación se realiza con la ayuda de la palanca de mando Space Navigator. Este controlador (también conocido como un ratón 3D) consta de varios ejes que permiten al usuario girar, empujar y tirar para volar sobre el globo virtual como si estuviera en un helicóptero.
Mediante programas o aplicaciones externas, el usuario puede interactuar con fotografías, vídeos, páginas web, etc. como superposiciones gráficas en el globo terráqueo para crear exposiciones e historias dinámicas. La instalación y uso de este sistema se encuentra en los documentos de su repositorio abierto oficial[13]

[12] Plano en el que el punto de vista de una cámara se encuentra perpendicular respecto del suelo y la imagen obtenida ofrece un campo de visión orientado de arriba abajo. [12]
[13] En la dirección web https://github.com/LiquidGalaxyLAB/liquid-galaxy

Figura 7: Palanca de comandos Space Navigator.

2.2.2. Google Earth

Google Earth es un programa informático que genera una representación 3D de la Tierra basada en imágenes de satélite. El programa mapea la Tierra mediante la superposición de imágenes obtenidas de fotografías satelitales y aéreas, y datos geográficos en un globo 3D. Earth es compatible con la gestión de datos geoespaciales tridimensionales mediante ficheros Keyhole Markup Language (KML). [14] Actualmente Google Earth tiene dos versiones disponibles:

- Google Earth Chrome: Versión web[14].
- Google Earth Pro: Versión local para profesionales. Permite la inserción de datos geográficos en el mapa, así como herramientas de medición, visualización de fotografías antiguas, etc.[15]

2.3. Base de Datos

La implementación de la base de datos del proyecto se puede realizar con el siguiente sistema:

2.3.1. MongoDB

MongoDB es una base de datos de documentos de código abierto que proporciona alto rendimiento, alta disponibilidad y escalado automático. Además, evita la necesidad de un Mapeo Relacional de Objetos (ORM) para facilitar el desarrollo. [15]

Un registro en MongoDB es un documento, una estructura de datos compuesta por pares de campo y valor. Los documentos MongoDB son similares a los objetos JSON. Los valores de los campos pueden incluir otros documentos, cadenas y cadenas de documentos. Los documentos son almacenados en colecciones, que son análogas a las tablas de las bases de datos relacionales. A diferencia de una tabla, sin embargo, una colección no requiere que sus documentos tengan el mismo esquema. En MongoDB, los documentos almacenados en una colección deben tener un único campo _id que actúe como clave principal. [16]

[14]Disponible en la web https://earth.google.com/web/

[15]Google Earth Pro requiere una descarga y instalación disponible en la web oficial de Google Earth https://www.google.com/earth/download/gep/agree.html

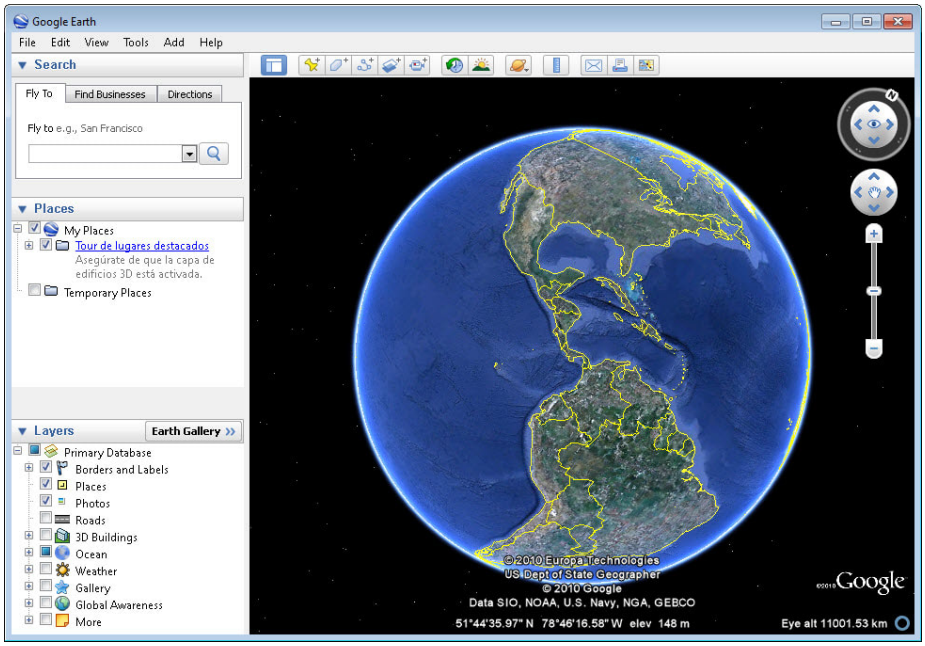


Figura 8: Programa Google Earth en funcionamiento.

Figura 9: Icono oficial de MongoDB.

Figura 10: Logo oficial de NodeJS.

Figura 11: Icono oficial npm.

2.4. Entornos de trabajo

La programación de las aplicaciones que componen el diseño de este proyecto debe realizarse con los entornos de trabajo mostrados a continuación:

2.4.1. NodeJS

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. [17]. Concebido como un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node está diseñado para construir aplicaciones en red escalables. Casi ninguna función en Node realiza I/O directamente, así que el proceso nunca se bloquea. Debido a que no hay bloqueo es muy razonable desarrollar sistemas escalables en Node. [18] Su implementación sencilla (sensores y imágenes fotográficas no contienen estructuras de datos complejas) y su potencial escalabilidad (los atributos de los sensores pueden variar dependiendo de su tipo o fabricante, así como el número de estos puede cambiar a deseo del cliente) hacen a Node.js el candidato ideal para la implementación del servidor del proyecto. Además, el ecosistema de paquetes de Node.js, npm, es el mas grande de librerías de código abierto.
Npm es un gestor de paquetes orientados a extender la funcionalidad de las aplicaciones Node, mediante la descarga o carga (importación) de librerías o módulos publicados en su repositorio oficial abierto[16]. Básicamente, npm se usa para la gestión de paquetes de Node (instalar, actualizar, eliminar y editar). Para poder utilizar un paquete, éste tiene que ser requerido ('importado') en

[16]Disponible en https://www.npmjs.com/

el archivo que se quiera utilizar en node, npm ve el paquete requerido por el archivo, y busca en su base de datos local del sistema para utilizarlo. Todos los paquetes y sus dependencias (paquetes necesitados de otros paquetes) se localizan dentro de un archivo llamado 'package.json', el cual puede contener también el nombre de la aplicación creada, una descripción y etiquetas para identificarla, entre otros campos necesarios para poder publicar dicha aplicación como un módulo / paquete más en el repositorio oficial.

Las aplicaciones implementadas con NodeJS se ejecutan a partir del fichero `package.json`[17], que contendrá todas las dependencias de módulos requeridos por la aplicación. Los archivos ejecutables en este entorno son escritos en lenguaje Javascript en ficheros con extensión `.js`, donde se pueden importar módulos centrales, llamados *core modules* o módulos externos publicados en el repositorio de npm, que deberan ser instalados previamente a la ejecución de los programas que los requieran.

2.4.2. ExpressJS

ExpressJS, descrito como Infraestructura web rápida, minimalista y flexible para Node.js [19], proporciona una gran ayuda a la programación del servidor, gracias a su gran número de métodos HTTP (llamadas GET, POST, etc.) y su incursión como middleware para crear una API robusta de forma rápida y simple. Para su uso, ExpressJS debe iniciar el servidor[18] indicando el número de puerto donde escuchar las conexiones. Una vez arrancado el servidor ya se pueden utilizar el gran abanico de métodos HTTP[19] de ExpressJS.

2.4.3. mongoose

Mongoose es otro módulo de NodeJS que proporciona una solución sencilla para la interacción con la base de datos MongoDB. Su funcionamiento está basado en esquemas para modelar los datos de MongoDB, y dispone de una gran variedad de funcionalidades que ayudan a la comunicación con la base de datos, incluyendo la validación, la construcción de consultas y la creación de valores. [20].

[17] El archivo `package.json` es un diccionario que define las características de la aplicación donde se sitúa. Puede contener campos como `name`, `version`, `description`, `author` y `license`, que son el nombre, versión, descripción, autor y licencia de la aplicación correspondiente. También puede incluir el campo 'main', que identifica el punto de entrada principal del programa, o el campo 'scripts', un diccionario que contiene los comandos que se ejecutan en varias ocasiones del ciclo de vida de la aplicación. `package.json` se puede generar automáticamente gracias a la llamada `npm init`.

[18] Mediante El método `app.listen(port, [hostname], [callback])`, que vincula y escucha las conexiones con el host (`hostname`) y el puerto (`port`) especificados, y ejecuta una función de devolución de llamada (`callback`).

[19] Vía la función `app.METHOD(path, callback)`, que dirige una solicitud HTTP, siendo `METHOD` el método HTTP de la solicitud, `path` la URL de esa solicitud, y `callback` la función de devolución de llamada a ejecutar. ExpressJS responde a un grupo muy extenso de métodos HTTP como `delete`, `get`, `head`, `options`, `patch`, `post` o `put`, entre otros

De forma similar al módulo ExpressJS, el funcionamiento de mongoose consiste primero en la conexión[20] a la base de datos MongoDB correspondiente, previamente iniciada.

Figura 12: Logo oficial de mongoose.

Mongoose trabaja con un modelo de datos que denomina `Schema`[21] (esquema). Cada esquema se asigna a una colección MongoDB y define la forma de los documentos dentro de esa colección. Un esquema puede tener muchas atributos, cada uno definido por un tipo diferente, delimitados a la siguiente lista:

- `String`: Cadena de caracteres
- `Number`: Número
- `Date`: Fecha
- `Buffer`: Datos binarios
- `Boolean`: Booleano.
- `Mixed`: Tipo indefinido (cualquier tipo)
- `ObjectId`: Identificador de objecto
- `Array`: Cadena / Colección

A partir de un esquema definido, mongoose lo compila[22] a un modelo `Model`. Las instancias de estos modelos representan documentos que se pueden guardar y recuperar de la base de datos.A través de estos mismos modelos, se pueden ejecutar diferentes métodos para interactuar con la base de datos. Los métodos más comunes son:

- `remove`: Elimina el documento de la base de datos.
- `save`: Guarda el documento.
- `create`: Guarda uno o mas documentos en la base de datos.
- `find`[23]: Busca documentos a partir de unas condiciones de búsqueda.

[20] La conexión con la base de datos se inicia con el método `mongoose.connect('mongodb://ruta-de-la-base-de-datos')`

[21] Dentro de un archivo Javascript, los esquemas se instancian a una variable de la forma `var nombreSchema = mongoose.Schema({ ...})`

[22] A través del código Javascript `var nombre = mongoose.model('nombre', nombreSchema)` donde instancia el modelo a una variable.

[23] El método `find` tiene varias variantes como `FindById` (Busca un único documento a partir de su atributo obligatorio '_id'.), `findByIdAndRemove` (Elimina el documento encontrado por findById.) y `findByIdAndUpdate` (Actualiza el documento encontrado por findById.).

Figura 13: Icono oficial Angular.

2.4.4. Angular

Angular (llamado también *Angular 2*), es un framework para aplicaciones web de TypeScript[24] de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. [22] Las aplicaciones Angular se crean mediante la composición de plantillas HTML con elementos 'angularizados'. El desarrollo de éstas aplicaciones se basa en las clases de los componentes que administran las plantillas, en la agregación de lógica de aplicación en servicios y en la agrupación de los componentes y los servicios en módulos. La aplicación se inicia mediante el arranque del módulo raíz. para que Angular tome el control, presentando el contenido de su aplicación en un navegador y respondiendo a las interacciones del usuario de acuerdo con las instrucciones que se le ha proporcionado.

Como se puede observar en la figura 14, los elementos principales de la arquitectura Angular son:

- Módulos (`Modules`): Angular tiene su propio sistema de modularidad llamado NgModules[25]. Todas las aplicaciones Angular tienen al menos una clase NgModule, el módulo raíz, denominado convencionalmente AppModule.

- Librerías: Angular se ejecuta como una colección de módulos JavaScript, similares a los módulos de una biblioteca. Cada librería Angular comienza con el prefijo @angular, se instala con el gestor de paquetes npm e importa partes de ellos con las instrucciones de importación de JavaScript.

[24] TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipado estático y objetos basados en clases. [21]

[25] NgModule es una clase adornada con la función del decorador @NgModule. @NgModule toma un objeto de meta-datos que indica a Angular cómo compilar y ejecutar código del módulo. Identifica los componentes propios del módulo, directivas y tuberías (pipes), haciendo que algunos de ellos sean públicos para que los componentes externos puedan usarlos. [23]

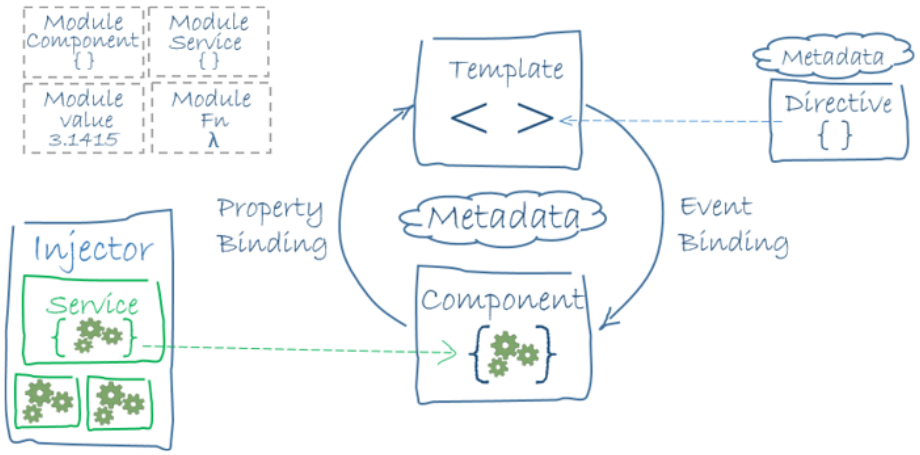


Figura 14: Diagrama general de la arquitectura de las aplicaciones Angular.

- Componentes (`Components`): Elemento que controla una sección de pantalla llamada vista (view). La lógica de un componente se define dentro de una clase, y ésta interactúa con la vista a través de una API de propiedades y métodos. Angular crea, actualiza y destruye componentes a medida que el usuario navega a través de la aplicación. La aplicación angular puede actuar en cada momento del ciclo de vida de un componente mediante un conjunto de métodos opcionales, son los mostrados en la figura 15.
- Plantillas (`Templates`): Forma de HTML que indica a Angular cómo renderizar un componente. Las plantillas son parecidas a HTML común, pero usan una "sintaxis de plantilla Angular" [24]
- Meta-datos (`Metadata`): Elemento que comunica a Angular cómo procesar una clase. En TypeScript, se adjuntan meta-datos utilizando un decorador, como puede ser '@Component', '@Injectable', '@Input' o '@Output', entre otros.
- Vinculación de datos (`Data binding`): Angular soporta la vinculación de datos[26], un mecanismo para coordinar partes de una plantilla con partes de un componente. Esta vinculación se realiza agregando ciertos elementos de enlace a la plantilla HTML para comunicar a Angular cómo conectar ambos lados.
- Directivas (`Directives`): Elemento que comunica las instrucciones que

[26] Existen cuatro formas de vinculación de datos sintaxis: Interpolación (`interpolation`), Vinculación de propiedad (`property binding`), Vinculación de eventos (`event binding`), y Vinculación bidireccional (`Two-way data binding`)

cumple el DOM[27] cuando Angular transforma una plantilla (en Angular las plantillas son dinámicas). Una directiva es una clase con el decorador @Directive. Un componente es una directiva adjunta a una plantilla, es decir, el decorador @Component es en realidad un decorador @Directive extendido con características orientadas a plantillas.

- Servicios (`Services`): Elemento que abarca cualquier valor, función o función que su aplicación necesita. Típicamente, un 'Service' es una clase con un propósito bien definido.
- Inyección de dependencia (`Dependency injection`): Forma para proporcionar una nueva instancia de una clase con las dependencias totalmente formadas que requiere. Generalmente las dependencias son servicios. Angular utiliza la inyección de dependencia para proporcionar nuevos componentes con los servicios que necesitan, mediante los parámetros del constructor del componente[28].

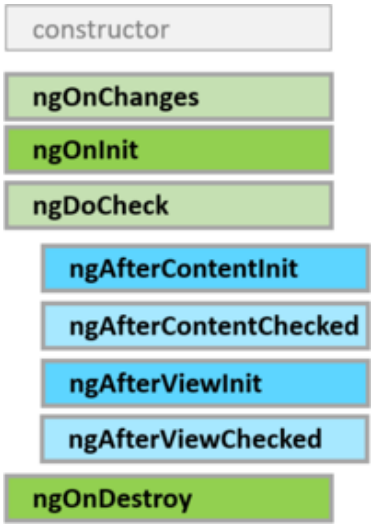


Figura 15: Ciclo de vida de un componente Angular

2.5. Lenguajes de programación

La funcionalidad de generar y enviar la información a un sistema Liquid Galaxy se tiene que producir con los siguientes lenguajes:

[27] "DOM (Document Object Model) es una interfaz de programación multiplataforma independiente de lenguaje que trata un documento HTML, XHTML o XML como una estructura de árbol, en la que cada nodo es un objeto que representa una parte del documento." [25]

[28] Mediante, por ejemplo: `constructor(private servicio: MiServicio){ ... }`

Figura 16: Icono oficial de Python.

2.5.1. Keyhole Markup Language (KML)

KML es un formato de archivo que se utiliza para mostrar datos geográficos en un navegador terrestre, como Google Earth, Google Maps y Google Maps para móviles. KML utiliza una estructura basada en etiquetas con atributos y elementos anidados y está basado en el estándar XML. Todas las etiquetas distinguen entre mayúsculas y minúsculas y deben aparecer exactamente como en las referencias oficiales de KML[29]. KML consta de la misma estructura dentro de sus ficheros[30], donde cada representación geográfica se añade con una etiqueta (`tag`) específica y con unos atributos determinados por su representación.

2.5.2. Python

El lenguaje Python se usa en los scripts generadores de KML del servidor. Python es un lenguaje de programación interpretado que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es administrado por la Python Software Foundation y posee una licencia de código abierto, denominada Python Software Foundation License, compatible con la Licencia pública general de GNU [26].

Los programas Python[31] pueden utilizar librerías externas mediante su importación[32]. Generalmente, estas librerías están compuestas de clases[33] que con-

[29] Disponibles en https://developers.google.com/kml/documentation/kmlreference

[30] Todos los ficheros KML que se quieran ejecutar en un sistema Liquid Galaxy deben incorporar estas líneas en su contenido: `<?xml version="1.0."en"coding=ÜTF-8¿><kml xmlns="http://www.opengis.net/kml/2.2» </kml>` donde las representaciones geográficas se colocan dentro de la etiqueta `kml`

[31] Los archivos ejecutables Python se guardan con la extensión `.py` y se ejecutan en el sistema mediante el comando `python nombre_del_programa.py`

[32] En Python Las librerías se importan con el código `import nombre_de_la_libreria`. También pueden importarse de forma individual una función de una librería mediante `from nombre_de_la_libreria import nombre_de_la_funcion`

[33] Python soporta una forma limitada de herencia múltiple en las clases, donde la función

tienen diversas funciones[34] que simplifican el proceso de programación en este lenguaje. La principal razón de la elección de Python para generar los ficheros KML, es la existencia de librerías que trabajan con este tipo de archivo, factor que ha simplificado de forma drástica la funcionalidad de generar estos ficheros.

2.5.3. simplekml

La librería Python simplekml tiene la función de simplificar el proceso de creación de representaciones geográficas en ficheros KML o KMZ[35]. Consta de una documentación completa y una variedad amplia de métodos para generar diferentes tipos de representaciones geográficas. Bajo licencia GNU Lesser General Public License[36], la simplicidad de su sintaxis respeto a otras librerías del mismo tipo (como por ejemplo PyKml) ha hecho decantar-se por su uso en este proyecto.

Simplekml[37] se basa en la construcción de objetos a través de una clase central[39]. La compilación del archivo KML se realiza a través de esta clase, ya que su base está asociada a un documento KML. Una vez instanciada la clase central, simplekml permite cambiar las propiedades del documento asociado[40], y es también a través de la misma clase que se crean las distintas representaciones geográficas[41] mediante los métodos definidos por simplekml, y finalmente se genera el fichero KML[42]. Simplekml puede generar en KML estas representaciones de entre una gran variedad:

- `point`: Ubicación geográfica con nombre y icono definida por longitud, latitud y altitud.
- `linestring`: Conjunto conectado de segmentos de línea definidos por unas coordenadas determinadas.
- `polygon`: Polígono definido por un límite exterior y / o un límite interior.
- `groundoverlay`: Superposición de imagen cubierta en el terreno.

`__init__` siempre se ejecuta al instanciar la clase.

[34] Las funciones en Python se declaran con `def funcion: ...`

[35] KMZ es un fichero KML que se encuentra comprimido con otros archivos, como por ejemplo una imagen.

[36] Licencia de software creada por la Free Software Foundation que pretende garantizar la libertad de compartir y modificar el software cubierto por ella, asegurando que el software es libre para todos sus usuarios. [27]

[37] Simplekml debe importarse[38] en el fichero Python donde se quiera ejecutar.

[39] Con simplekml cualquier acción empieza con la creación de una instancia de la clase simplekml.Kml de la forma `kml = simplekml.Kml()`

[40] Para cambiar las propiedades después de la creación, se realiza a través de la propiedad simplekml.Kml.document (). Por ejemplo, para cambiar el nombre del documento kml:`kml.document.name = "Nuevo nombre"`

[41] Simplekml crea figuras con representaciones geográficas a través de la instancia del objeto simplekml.Kml de la forma: `kml.newTIPO_DE_REPRESENTACION()` siendo TIPO_DE_REPRESENTACION la representación geográfica deseada de entre el catálogo que contiene la librería.

[42] l guardado del fichero KML se realiza con el método `kml.save("NOMBRE_DEL_FICHERO_KML")`

3. Análisis y diseño

3.1. Análisis de requerimientos

Para un diseño que funcione correctamente a las necesidades de los usuarios, es recomendable realizar un análisis de los requerimientos que va a tener el proyecto, es decir, las cualidades que tendría que cumplir el programa una vez esté finalizado su desarrollo. El listado de estos requerimientos se ha distribuido en dos grupos: requerimientos funcionales, aquellos que corresponden a las funcionalidades que ha de disponer el sistema o aplicación a desarrollar, y los no funcionales, relacionados con características y cualidades de aspecto más general necesarias de una aplicación.

3.1.1. Requerimientos funcionales

- La aplicación debe encargarse del almacenamiento de datos relacionados con sensores y fotografías de drones.
- La aplicación debe permitir toda subida de imágenes deseada por el usuario.
- La aplicación debe encargarse del automatismo en la obtención de los datos de una imagen en el momento de su subida.
- La aplicación debe permitir al usuario poder visualizar cualquier dato de sensor guardado en el servidor.
- La aplicación debe permitir al usuario poder visualizar cualquier imagen guardada en el servidor.
- La aplicación debe permitir la interacción entre el sistema y el usuario para que éste pueda enviar por sí mismo información a un sistema Liquid Galaxy.
- La aplicación debe permitir una vía simple y rápida para el envío de datos al Liquid Galaxy que sirva de demostración de las representaciones geográficas del proyecto.

3.1.2. Requerimientos no funcionales

- La aplicación ha de tener suficiente capacidad de almacenaje para poder guardar una cantidad alta de valores de sensores e imágenes.
- La interacción entre la aplicación y el usuario debe ser simple e intuitiva, sin necesidad de ayuda externa.
- La interfaz de la aplicación debe mantener una similitud visual y un orden parecido entre todas sus páginas para mejorar la experiencia de usuario y evitar la confusión.

- El sistema de subida de datos de la aplicación debe ser estable y con el tiempo de respuesta más corto posible.

3.2. Diseño general

El diseño del proyecto está compuesto por tres aplicaciones:

- Servidor de datos `sensor-api`: Aplicación NodeJS que realiza la función de servidor de almacenaje de datos con una base de datos mongoDB.
- Servidor de ficheros KML `kml-server`: Aplicación NodeJS que ejecuta el proceso de generación y envío de archivos KML a partir de pequeños programas Python.
- Cliente web `SAVT-Dashboard`: Aplicación Angular que tiene como función la visualización de datos a través de un navegador web y la interacción con el usuario para el envío de información a un sistema Liquid Galaxy.

Como se puede observar en el diagrama de la figura 17, mostrada seguidamente, la interacción entre las aplicaciones se inicia en el momento en que el usuario arranca (o más bien visita) el contenido del cliente `SAVT-Dashboard`. En ese instante, el mismo cliente realiza llamadas HTTP al servidor `sensor-api` para poder mostrar en el navegador los datos que el usuario ha solicitado, dependiendo de la página a la que ha navegado. Si el usuario decide visualizar la información en un sistema Liquid Galaxy a través de la interfaz, el cliente lanza una determinada llamada HTTP al servidor `kml-server`, para que éste genere las representaciones geográficas con la información seleccionada por el usuario, y la envíe finalmente al equipo Liquid Galaxy.

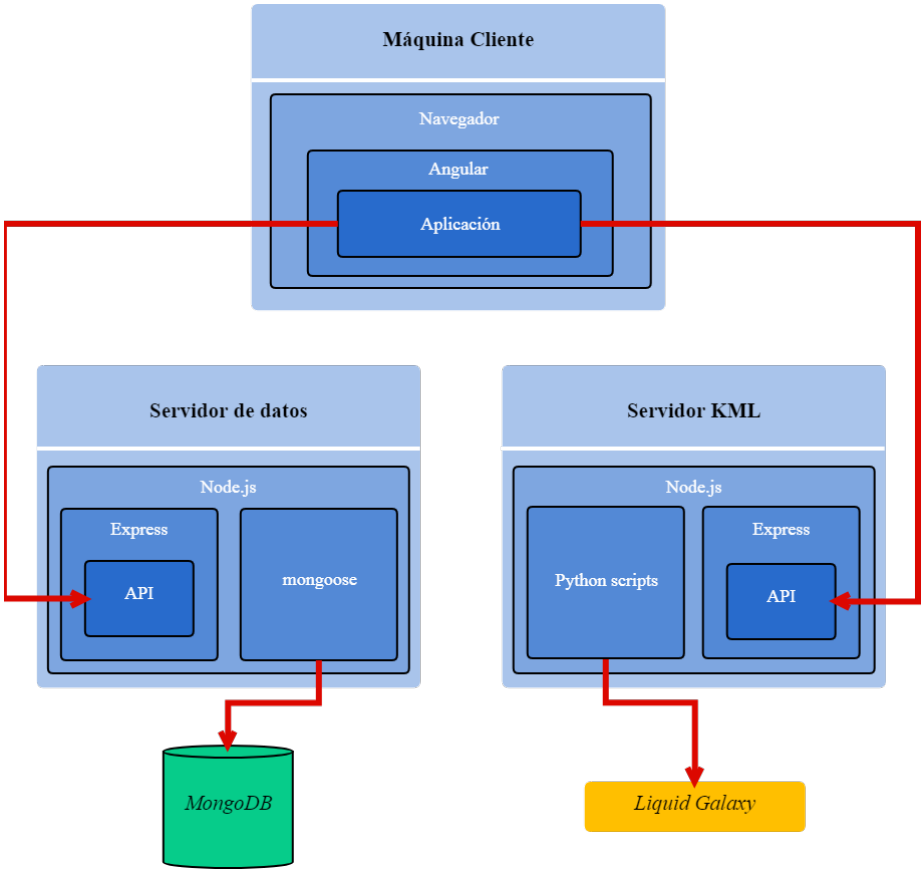


Figura 17: Diseño general del proyecto.

3.3. Modelo de datos

El modelo de datos diseñado para el servidor de datos del proyecto se ha creado con vistas a organizar su contenido en dos tipos de información: **sensores** e **imágenes**, como se puede contemplar en la figura 18.
La información de **sensores** se distribuye con el modelo `Field`, que es un campo de cultivo que contiene un numero determinado de sensores, representados por el modelo `Sensor`. Las **imágenes**, por su parte, están organizadas en cuatro modelos distintos: Por una parte, se ha diseñado el modelo `Author` que representa a un creador (normalmente una empresa) de fotografías de drones. Este creador contiene un número determinado de álbumes de fotografías, representados por el modelo `Album`. Y cada álbum contiene también un número de imágenes, cada una interpretada con el modelo `Image`. Por otro lado, el modelo `Overlay` es similar a un álbum de fotografías, pero que sólo contiene imágenes de la misma localización (con diferentes análisis). Se ha diseñado este ultimo modelo para poder generar la información en los archivos KML de una forma más simple.

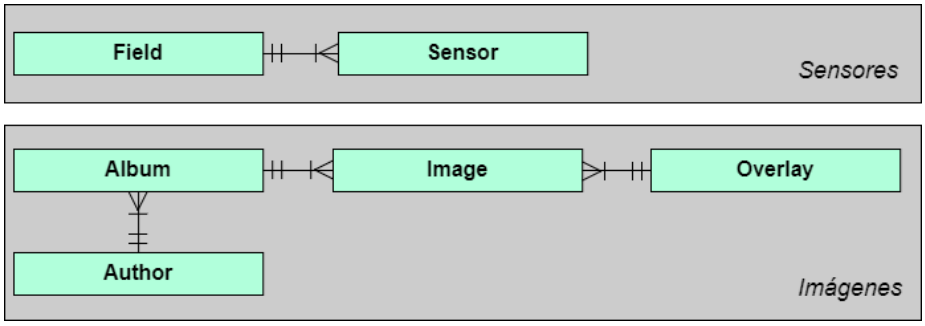


Figura 18: Diseño general de la base de datos

A continuación se definen individualmente los seis modelos de datos diseñados:

- `Field` (figura 19): El modelo Field equivale a un Campo de cultivo determinado. Cada campo debe contener al menos un sensor comprendido en el atributo `sensors`, una cadena de identificadores (id) únicos de sensores.

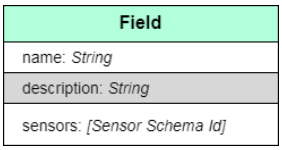


Figura 19: Tabla Field (Campo).

- `Sensor` (figura 20): La tabla Sensor abarca un sensor en una posición determinada (los atributos `locationLatitude` (latitud) y `locationLongitude` (longitud) son obligatorios) con un conjunto de atributos opcionales que representan los valores de las magnitudes que el sensor está midiendo.

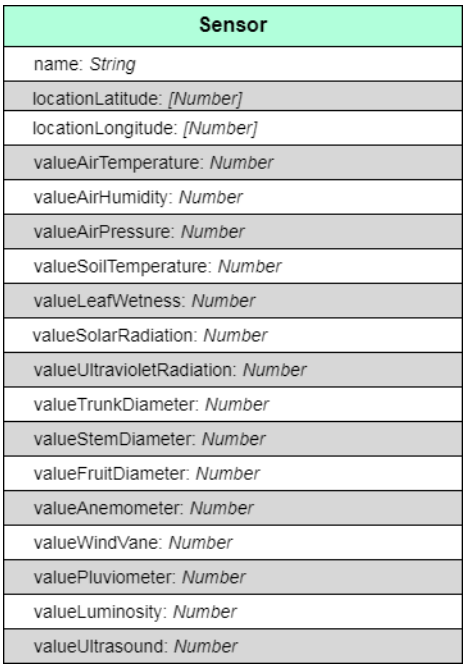


Figura 20: Tabla Sensor (Sensor).

- `Author` (figura 21): Author representa a un Autor o propietario de unas imágenes de drones. Puede contener diversos álbumes (tabla Album) gracias al atributo `albums`, que guarda una cadena los identificadores (id) de esos álbumes en la base de datos.

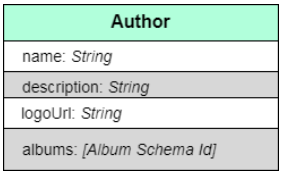


Figura 21: Tabla Author (Autor).

- `Album` (figura 22): El campo Album corresponde a un álbum de imágenes. Cada álbum una cadena de los identificadores (id) de las imágenes que de ésa colección guardada en el atributo `images`.

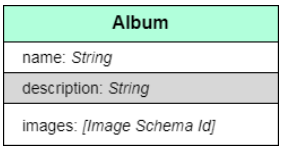


Figura 22: Tabla Album (Álbum).

- `Image` (figura 23): Image representa una imagen guardada en la base de datos con las coordenadas de la posición donde ha sido tomada.

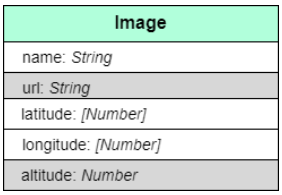


Figura 23: Tabla Image (Imagen).

- `Overlay` (figura 24): El tipo Overlay (superposición) identifica las imágenes que son de un mismo sitio (que contienen las mismas coordenadas de localización) pero que no son la misma imagen (la misma fotografía pero con cambios visuales como su análisis), y por tanto con diferente nombre. Una superposición debe contener mínimamente una imagen en su conjunto del atributo `images`. Además, Overlay contiene los atributos `markerDL`, `markerDR`, `markerUR` y `markerUL`, que corresponden a las coordenadas de cada una de las esquinas (inferior izquierda, inferior derecha, superior derecha y superior izquierda, respectivamente) de la imagen, valores necesarios para la inserción de la imagen como superposición de suelo en un sistema Liquid Galaxy.

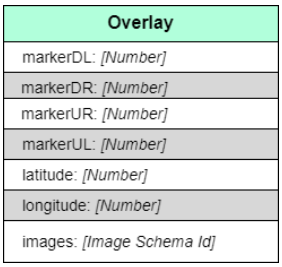


Figura 24: Tabla Overlay (Superposición).

La siguiente sección contiene una explicación de los componentes y funcionalidades que se han programado en la creación de las aplicaciones del proyecto.

4. Implementación

A continuación se listan los procedimientos que han de seguirse para instalar cada una de las aplicaciones del proyecto.

4.1. Instalación y configuración

4.1.1. MongoDB

MongoDB debe descargarse desde su página oficial de instalación[43]. Una vez descargado, debe instalarse siguiendo los pasos descritos en el manual dependiendo de la versión y sistema al que se quiera instalar. En un sistema Linux, MongoDB puede iniciar la base de datos con el comando:

```
$ mongod --dbpath database_path
```

Siendo `database_path` la ruta hacia el directorio donde se quiera guardar o se hayan guardado los componentes de la base de datos.

4.1.2. NodeJS

La forma más común de instalar node.js es compilarlo directamente desde el código fuente descargado desde su página oficial 'nodejs.org'. En la terminal de un sistema operativo Linux se ejecutarían los siguientes comandos: Descarga del código fuente (siendo `X` la versión a escoger):

```
$ wget http://nodejs.org/dist/node-vX.X.X.tar.gz
```

Descompresion del paquete descargado:

```
$ tar -xzf node-vX.X.X.tar.gz
```

Navegación y ejecución de los scripts de instalación:

```
$ cd node-v0.4.4
$ ./configure
$ sudo make install
```

Dado que Node.js no tiene dependencias externas excepto las herramientas de compilación comunes, es probable que se necesite ejecutar:

```
$ apt-get -y install build-essential
```

Como parte de Node, el gestor de paquetes npm también se instalará en el sistema. Para comprobar que la instalación se ha realizado de forma correcta se puede ejecutar la orden para obtener la versión de cada programa:

[43] Disponible en su web oficial de instalación: https://docs.mongodb.com/manual/installation/

```
$ node -v
$ npm -v
```

4.1.3. Express

Habiendo inciado previamente un proyecto Node.js, se instala Express ejecutando el siguiente comando dentro del directorio raíz:

```
$ npm install express
```

4.1.4. mongoose

Habiendo incializado previamente un proyecto Node.js, mongoose se puede instalar con el siguiente comando ejecutado dentro del directorio raíz:

```
$ npm install mongoose
```

4.1.5. Angular

Para poder crear aplicaciones Angular es necesario tener instalado previamente los entornos:

- node: Versión mínima 6.9.x
- npm: Versión mínima 3.x.x

La manera más recomendable de crear un proyecto Angular es a través de la interfaz de línea de comandos Angular CLI. "Angular CLI es una herramienta para inicializar, desarrollar, escalar y mantener aplicaciones angulares" [28]. Para instalar Angular CLI de forma global:

```
npm install -g @angular/cli
```

Con Angular CLI instalado en el sistema, se genera un nuevo proyecto con una estructura predefinida con el comando:

```
ng new nombre-del-proyecto
```

La aplicación Angular vive en la carpeta src. Todos los componentes, servicios, plantillas, estilos, imágenes, etc. se guardan en este directorio, teniendo la siguiente estructura inicialmente:

```
src
 | ----- app
 |         | ----- app.component.css
 |         | ----- app.component.html
 |         | ----- app.component.spec.ts
 |         | ----- app.component.ts
 |         | ----- app.module.ts
 | ----- assets
 |         | ----- .gitkeep
 | ----- environments
 |         | ----- environment.prod.ts
 |         | ----- environment.ts
 | ----- favicon.ico
 | ----- index.html
 | ----- main.ts
 | ----- polyfills.ts
 | ----- styles.css
 | ----- test.ts
 | ----- tsconfig.app.json
 | ----- tsconfig.spec.json
```

Dónde cada fichero tiene una función diferente dentro de la aplicación:

- `app/app.component.ts, html, css, spec.ts`: Define AppComponent junto con una plantilla HTML, una hoja de estilo CSS y test unitario. Es el componente raíz de lo que se convertirá en un árbol de componentes anidados a medida que la aplicación evolucione.
- `app/app.module.ts`: Define AppModule, el módulo raíz que indica a Angular cómo construir la aplicación. Inicialmente sólo declara AppComponent, y si la aplicación contiene más componentes, deberá declararlos en este fichero.
- `assets/*`: Directorio para guardar elementos (por ejemplo imágenes) que se copiaran completamente cuando se construya la aplicación.
- `environments/*`: Directorio que contiene un archivo para cada uno de los entornos de destino, cada uno de los cuales exporta variables de configuración simples para usar en su aplicación.
- `favicon.ico`: Icono de página (visible usualmente en la barra / pestaña del navegador) de la aplicación. Inicialmente es el icono oficial de Angular.
- `index.html`: Página HTML principal que se sirve cuando se abre la aplicación. Inicialmente Angular CLI añade diversos archivos js y css al crear la aplicación, para acelerar el proceso de desarrollo.

- `main.ts`: Punto de entrada principal de la aplicación. Compila la aplicación con el compilador JIT[44] y arranca el módulo raíz de la aplicación (AppModule) para ejecutarse en el navegador.
- `polyfills.ts`: Polyfills ayudan a normalizar los diferentes niveles de soporte de los estándares web. style.css: Fichero central que guarda los estilos que afectan de forma global a todos los componentes de la aplicación.
- `test.ts`: Punto de entrada principal para los test unitarios.
- `tsconfig.app|spec.json`: Configuración del compilador TypeScript para la aplicación Angular (tsconfig.app.json) y para los test unitarios (tsconfig.spec.json).

Una vez creado, desde su directorio raíz se puede servir dicho proyecto para visualizar su resultado en un navegador con:

```
ng serve
```

Durante el desarrollo de la aplicación, Angular CLI puede agilizar el proceso mediante sus diferentes métodos. El más destacado es la creación automática de elementos del proyecto con la orden:

```
ng generate tipo nombre_del_elemento
```

Siendo 'tipo' el elemento a crear, como `component` o `service`, entre otros.

4.1.6. Python

Hoy en día muchas distribuciones de Linux y UNIX incluyen una versión de Python reciente, por lo que es recomendable comprobar primero que no se tiene ya instalado mediante la simple orden `python`. Si se necesita instalar, se debe descargar en la web oficial[45] y instalarlo según las instrucciones que estén dentro del paquete comprimido.

4.1.7. simplekml

simplekml debe descargarse a través del repositorio oficial de paquetes python[46] y instalarse segun los pasos descritos en su página oficial. El paquete tambien puede instalarse a través del sistema de gestión de paquetes de Python *pip* con el comando:

```
$ pip install simplekml
```

[44] "Compilación Just-in-Time (JIT). Un método bootstrap para compilar componentes y módulos en el navegador y lanzar la aplicación dinámicamente." [29]
[45] En la dirección https://www.python.org/downloads/
[46] En la página https://pypi.python.org/pypi/simplekml

4.2. Realización

La realización del proyecto se ha distribuido en la implementación de las tres aplicaciones que lo componen: servidor de datos (nombrado `sensor-api`), servidor de generación y envío de KMLs (denominado `kml-server`) y cliente web (llamado `SAVT-Dashboard`).

En la primera parte de este apartado se explica la implementación de los dos servidores respecto a los métodos que contienen cada uno, para sus respectivas conexiones con el cliente.

4.2.1. Servidor de datos (sensor-api)

El servidor `sensor-api` tiene como función almacenar y servir los datos de sensores y imágenes, por lo que ha sido necesario crear un conjunto de llamadas de forma similar a las de un servidor API, que permitan la creación, modificación y eliminación de cada campo de los modelos de la base de datos. El grupo de llamadas implementado en `sensor-api` se muestra en las figuras 25 y 26[47]., mostradas seguidamente. A continuación se lista en detalle la información técnica del conjunto de llamadas HTTP desarrolladas para el servidor sensor-api. Las llamadas han estado separadas por los dos tipos de datos que se muestran en la aplicación: sensores, que corresponden al diagrama de la figura 25, y imágenes, mostrados en la figura 26. En cada tipo de dato se define individualmente cada método[48] de la ruta donde corresponde, donde cada ruta pertenece a un modelo diferente diseñado en la base de datos. En los datos de sensores las llamadas son las siguientes:

- Ruta /fields
  - GET /fields/ Retorna la lista de campos guardados en la base de datos.
  - GET /fields/:id Retorna los detalles de un campo definido con su identificador ':id'.
  - POST /fields/ Crea un nuevo campo en la base de datos con los valores introducidos en el cuerpo ('body') JSON de la llamada HTTP. Dicho cuerpo debe contener obligadamente un nombre único del campo en el atributo 'name', y al menos un identificador de un sensor de la base datos en el atributo 'sensors'.
  - DELETE /fields/all Elimina todos los campos guardados en la base de datos.

[47]Los tres diagramas de las figuras 25, 26 y 27 muestran la información distribuida en diferentes colores: *Gris* (las rutas de cada modelo de datos), *Verde* (llamada `GET`. Corresponde a una obtención de datos), *Naranja* (llamada `POST`. Creación y modificación), *Azul* (llamada `PUT`. Exclusiva para la modificación de datos) y *Rojo* (llamada `DELETE`. Pertenece a la eliminación de los datos.)

[48]Téngase en cuenta que todas las llamadas a los métodos `POST` de creación y todos los `DELETE`, de eliminación, se han creado para el desarrollo de la aplicación y no son llamados por el cliente del proyecto.

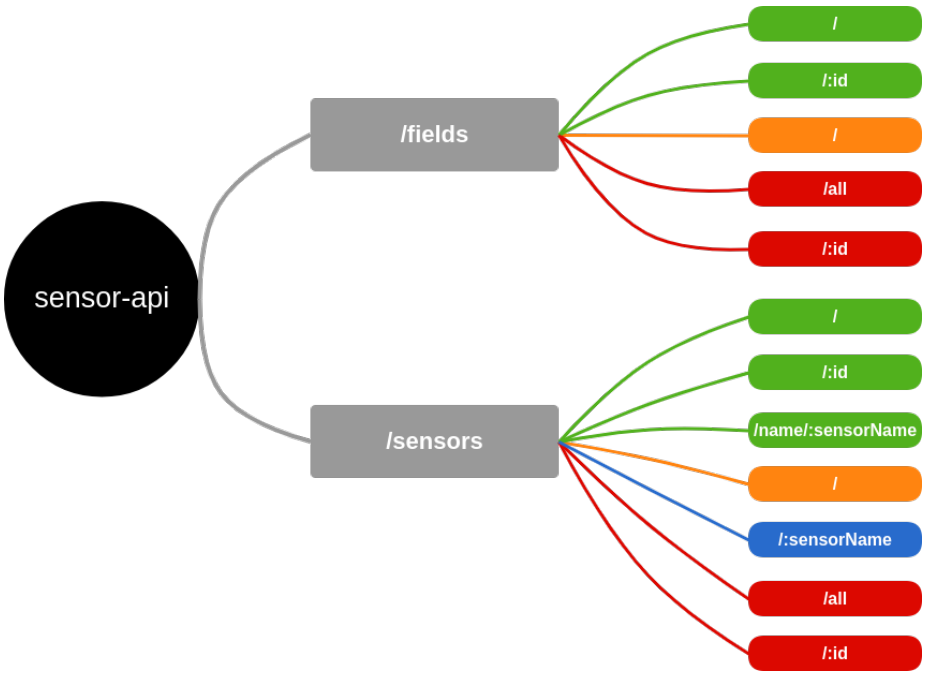


Figura 25: Diagrama de las llamadas a métodos HTTP de las rutas relacionadas con el tipo de datos sensores.

    - DELETE /fields/:id Elimina un campo de la base de datos determinado por su identificador en el parámetro ':id'.

- Ruta /sensors
    - GET /sensors/ Retorna la lista de sensores (sensors) guardada en la base de datos. Además, se han desarrollado varias funciones Javascript en ésta llamada para que, junto a la lista de sensores, se devuelva un resumen (summary) del estado general de los sensores. El resumen separa los sensores disponibles ('available', que miden un valor en alguna de las magnitudes) de los no disponibles ('unavailable', que no contienen valores guardados en ninguna magnitud), y enumera de forma similar las magnitudes disponibles ('available') de las que no ('unavailable'). 'Summary' se ha desarrollado con al intención de evitar ciertas operaciones de búsqueda y listado al cliente en la implementación de futuras páginas de visualización de datos.
    - GET /sensors/:id Retorna los detalles (todos sus atributos) del sensor con identificador ':id'.
    - GET /sensors/name/:sensorName Retorna los detalles de un sensor determinado a través de su nombre (su campo 'name') con el parámetro ':sensorName'. Éste método es llamado por el cliente en las situaciones en que sólo consta del nombre del sensor.
    - POST /sensors/ Crea un nuevo sensor en la base de datos con los valores escritos en el cuerpo (body) JSON de la llamada HTTP. 'Body' debe contener obligadamente un nombre único del sensor con la clave 'name', y las coordenadas de su localización con las claves 'locationLatitude' (latitud) y 'locationLongitude' (longitud).
    - PUT /sensors/:sensorName Modifica los valores de un sensor determinado por su nombre ':sensorName'. Se ha desarrollado éste método con vistas a posibles integraciones del proyecto en casos reales con sensores, para que el servidor 'sensor-api' pueda modificar los valores medidos por los mismos sensores a través de ésta llamada.
    - DELETE /sensors/all Elimina todos los sensores guardados en la base de datos.
    - DELETE /sensors/:id Elimina un sensor de la base de datos determinado por su identificador en el parámetro ':id'.

Respecto a los datos relacionados con las imágenes, las llamadas se definen en la siguiente lista:

- Ruta /authors
    - GET /authors/ Retorna la lista de autores (authors) guardados en la base de datos.
    - GET /authors/:id Retorna los detalles de un autor (author) referido con su identificador ':id'.

- GET /authors/name/:authorName Retorna los detalles de un sensor buscado a través de su nombre (‘name’) ‘:authorName’. Éste método se llama en el cliente de la misma forma que en GET /sensors/name/:sensorName.
- POST /authors/ Crea un nuevo autor en la base de datos con los valores del cuerpo (‘body’) JSON de la llamada, dónde son requeridos un nombre (‘name’) único y al menos un identificador (id) de álbum en el campo ‘albums’.
- DELETE /authors/all Elimina todos los autores guardados en la base de datos.
- DELETE /authors/:id Elimina un autor de la base de datos determinado por su identificador en el parámetro ‘:id’.

- Ruta /albums
  - GET /albums/ Retorna la lista de álbumes guardados en la base de datos.
  - GET /albums/:id Retorna los detalles de un álbum definido con su identificador ‘:id’.
  - POST /albums/ Crea un nuevo álbum en la base de datos con los valores del cuerpo (‘body’) JSON de la llamada. En su creación se require un nombre único de álbum y al menos un identificador de imagen en sus respectivas claves dentro del cuerpo nombrado.
  - POST /albums/upload/:albumName Añade una imagen nueva dentro del álbum con nombre ‘:albumName’. Esta imagen es añadida gracias a su identificador en el campo ‘imageId’ del cuerpo de la llamada.
  - DELETE /albums/all Elimina todos los álbumes guardados en la base de datos.
  - DELETE /albums/:id Elimina un álbum de la base de datos determinado por su identificador en el parámetro ‘:id’.

- Ruta /images
  - GET /images/ Retorna la lista de imágenes guardadas en la base de datos.
  - GET /images/:id Retorna los detalles de una imagen definida con su identificador ‘:id’.
  - POST /images/ Crea una nueva imagen en la base de datos. Éste método no requiere información explícita en el cuerpo de la llamada, ya que contiene la implementación de un código que automatiza la inserción de sus valores: al subir una imagen desde el cliente y llamar al método, se obtiene el nombre, latitud y longitud de la imagen a

través de sus datos Exif[49], y se añaden al 'body' junto al campo 'url', que se genera con la concatenación de la dirección ip donde se ejecuta el servidor y el nombre de fichero generado por la librería multer[50].

- DELETE /images/all Elimina todas las imágenes guardadas en la base de datos.
- DELETE /images/:id Elimina una imagen de la base de datos determinada por su identificador en el parámetro ':id'.

- Ruta /overlays
  - GET /overlays/ Retorna la lista de superposiciones guardadas en la base de datos.
  - GET /overlays/:id Retorna los detalles de una superposición definida con su identificador ':id'.
  - GET /overlays/image/:imageId Retorna los detalles de una superposición que contiene en su campo 'images' un identificador de imagen igual al parámetro ':imageId'. Éste método es usado por el cliente para verificar si una imagen ya está fichada dentro de una superposición determinada.
  - POST /overlays/ Crea una nueva superposición en la base de datos con los valores introducidos en el cuerpo ('body') JSON de la llamada HTTP. Dicho cuerpo debe contener obligadamente las coordenadas (latitud en el campo 'latitude' y longitud en el campo 'longitude') de la superposición y al menos un identificador de imagen en el campo 'images'.
  - POST overlays/saveMarkers/image/:imageId Guarda las nuevas coordenadas de las cuatro esquinas de la superposición (definidas en el 'body' de la llamada) de la superposición que contenga en su cadena 'images' el identificador de imagen ':imageId'.
  - POST overlays/uploadImage/image/:imageId Añade el nuevo identificador de imagen ':imageId' a la cadena 'images' de la superposición con las coordenadas (latitud y longitud) definidas en el 'body' JSON de la llamada.
  - DELETE /overlays/all Elimina todas las superposiciones guardadas en la base de datos.
  - DELETE /overlays/:id Elimina una superposición de la base de datos determinada por su identificador en el parámetro ':id'.

[49] Exif, Formato de archivo de imagen intercambiable, es un estándar que especifica los formatos para imágenes, sonido y etiquetas auxiliares usadas por cámaras digitales, escáneres y otros sistemas. Ésta especificación usa el archivo de formato existente y le agrega etiquetas (tags) específicos de meta-datos, como información sobre la localización de la imagen, que usualmente proviene de un GPS conectado a la cámara. [30]

[50] Middleweare node.js para la interacción con datos multipart / form-data, usados principalmente para la subida de archivos. [31]

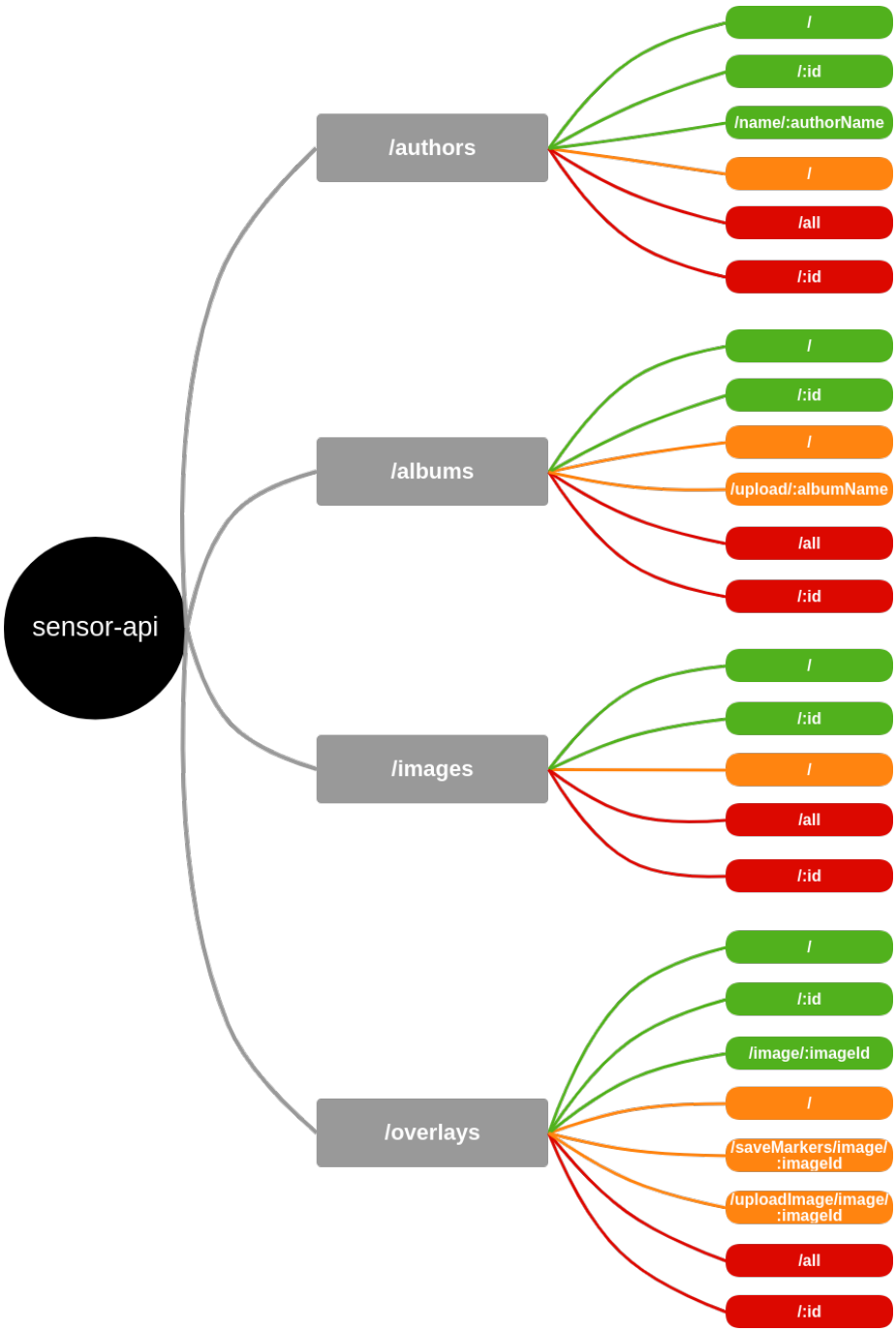


Figura 26: Diagrama de las llamadas a métodos HTTP de las rutas relacionadas con el tipo de datos imágenes.

4.2.2. Servidor de KML (server-kml)

Para conseguir una comunicación entre el cliente y un sistema Liquid Galaxy, ha sido necesario desarrollar un nuevo servidor, que debe ser ejecutado en un ordenador que esté dentro de la misma red que el Liquid Galaxy al que se quiere enviar información, ya que actualmente el paso de los ficheros KML se realiza a través de SSH de forma local. El servidor `kml-server` tiene exclusivamente como función la generación y envío de archivos KML. La inserción de valores en el proceso de creación de dichos archivos KML se produce gracias a la información que recibe en el cuerpo (`body`) de las llamadas HTTP, por lo que no es necesario el almacenamiento de ningún tipo de dato, y por tanto no se ha diseñado una nueva base de datos para éste servidor.
A continuación se muestran el abanico de llamadas HTTP, visibles en el diagrama de la figura 27, que han sido implementadas en el servidor kml-server:

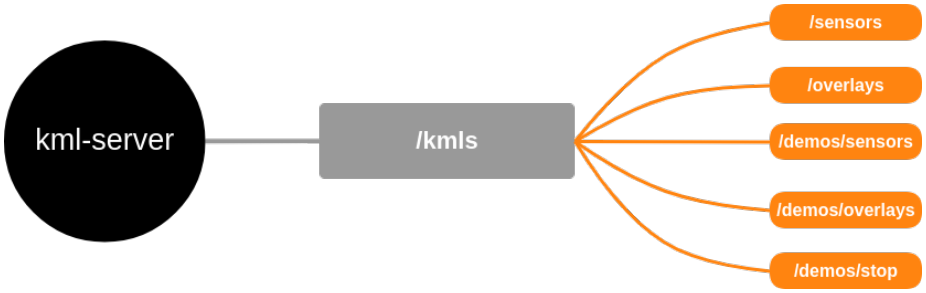


Figura 27: Diagrama de las llamadas a métodos HTTP al servidor de KML sensor-kml.

- POST kmls/sensors Ordena la generación del fichero KML con los datos de los sensores seleccionados por el usuario y el posterior envío al sistema Liquid Galaxy.
- POST /overlays Ordena la generación del fichero KML con las imágenes (superposiciones) seleccionadas por el usuario y el posterior envío al sistema Liquid Galaxy.
- POST /demos/sensors Ordena el envío del archivo KML pre-generado de demostración de datos de sensores (guardado en el servidor kml-server) al sistema Liquid Galaxy.
- POST /demos/overlays Ordena el envío del archivo KML pre-generado de demostración de superposición de imágenes (guardado en el servidor kml-server) al sistema Liquid Galaxy.
- POST /demos/stop Ordena la parada de la visualización de datos del sistema Liquid Galaxy.

4.2.3. Cliente (SAVT-Dashboard)

El cliente Angular llamado 'SAVT-Dashboard' (panel de datos Smart Agro Visualization Tool) es la principal aplicación de éste proyecto, dado que hace de conector entre todas las demás. Además, es la única aplicación visible al usuario final, factor que conlleva desarrollar vistas hacia el usuario. El diagrama de la figura 28 muestra una vista general de la estructura que se ha creado en el desarrollo de la aplicación cliente SAVT-Dashboard. Como se puede observar, la aplicación parte del componente 'App Component', que define en su plantilla el componente Sidenav (menú lateral de navegación de Angular Material) junto a Toolbar (componente Material que define una barra de herramientas) y el componente Router-Outlet. Router-Outlet es un componente original de Angular que renderiza otros componentes según la ruta (URL) visitada en el momento. Ésta implementación en el componente raíz de la aplicación hace que tanto el menú lateral como la barra de herramientas estén generadas en cualquier página mientras se navega. Cabe señalar que los componentes Sidenav y Toolbar tiene una comunicación con el servicio 'Sidenav Service', creado explícitamente para permitir una interacción entre estos dos componentes (en este caso, poder abrir y cerrar el menú desde la barra de herramientas). En SAVT-Dashboard se puede navegar a cuatro páginas diferentes, que corresponden cada una a un componente distinto: kml-generator (página principal), sensors-page (visualización de sensores), gallery-page (galería de fotografías) y demos-page (demostraciones).

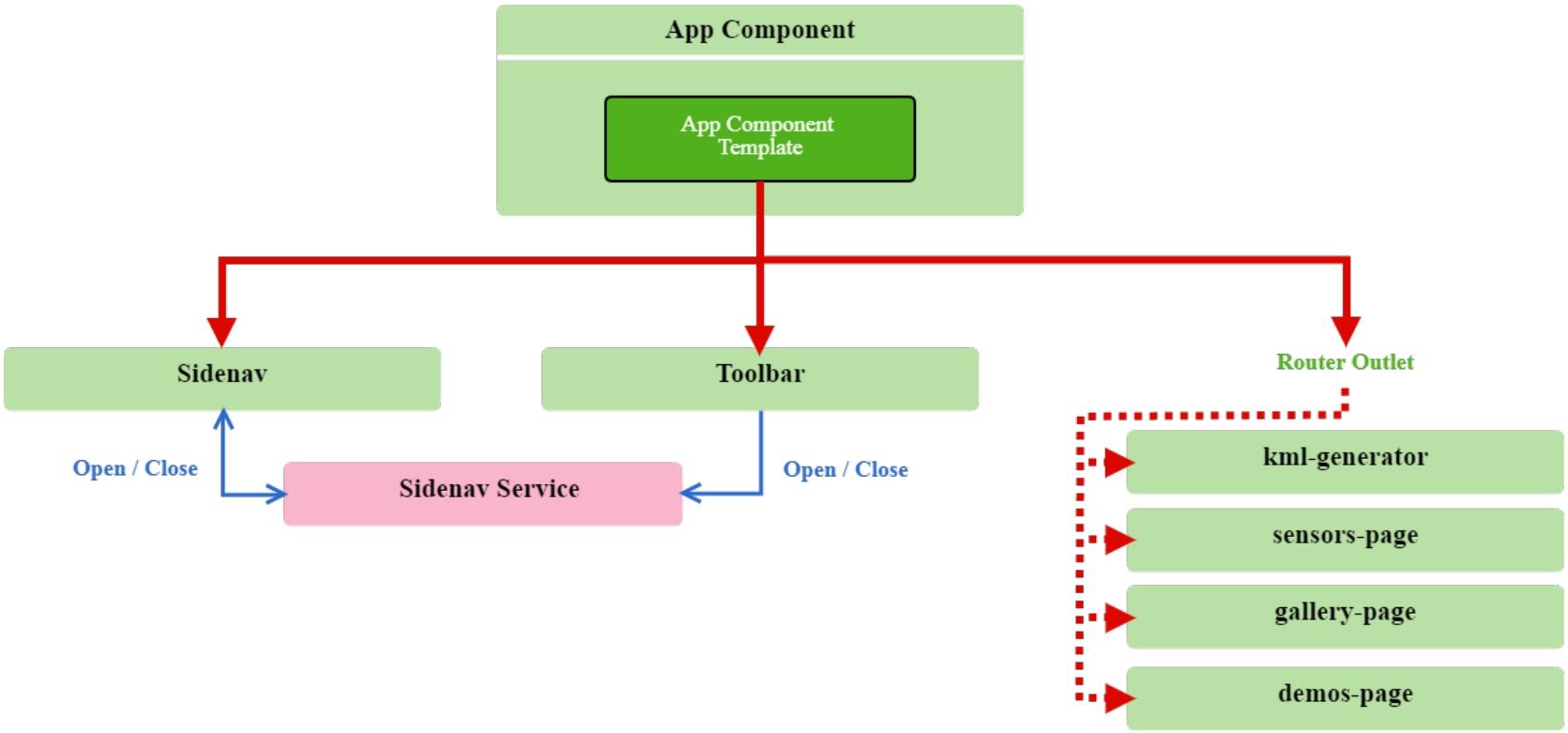


Figura 28: Diagrama de la estructura general de la aplicación Angular SAVT-Dashboard.

A continuación se describen los componentes y su contenido, así como la interacción que tienen con el resto de elementos de la aplicación:

- `kml-generator`: El componente kml-generator, en la ruta '/kml', representa la página principal de la aplicación. Es el componente con la estructura más compleja, ya que en su interior contiene tres componentes diferentes más el contenido de su propia plantilla. En dicha plantilla, se muestran datos de sensores que son obtenidos mediante el servicio 'Sensor Service', el cual es el responsable de conectar con el servidor de datos 'sensor-api' con las llamadas HTTP listadas antes. Además, 'kml-generator' visualiza los datos relacionados con las imágenes guardadas en el servidor, a través del servicio 'Image Service', que también contiene las llamadas HTTP nombradas. En ésta página, cuando es seleccionado algún valor de estos dos tipos de datos, se comunica a respectivo servicio (los datos de sensores con 'Sensor Service' y los datos de imágenes con 'Image Service') para visibilizar su información en el mapa virtual generado por el componente 'google-map'. Por último, también en la misma plantilla de 'kml-generator', se incluye una opción (botón para crear y enviar un KML) que al seleccionarla, se despliega un nuevo componente: 'dialog', una ventana que se superpone al contenido de la página y, en este caso, comunica con métodos tanto de 'Image Service' como de 'Sensor Service' para llamar a algunas funciones del servidor 'kml-server' comentado anteriormente.

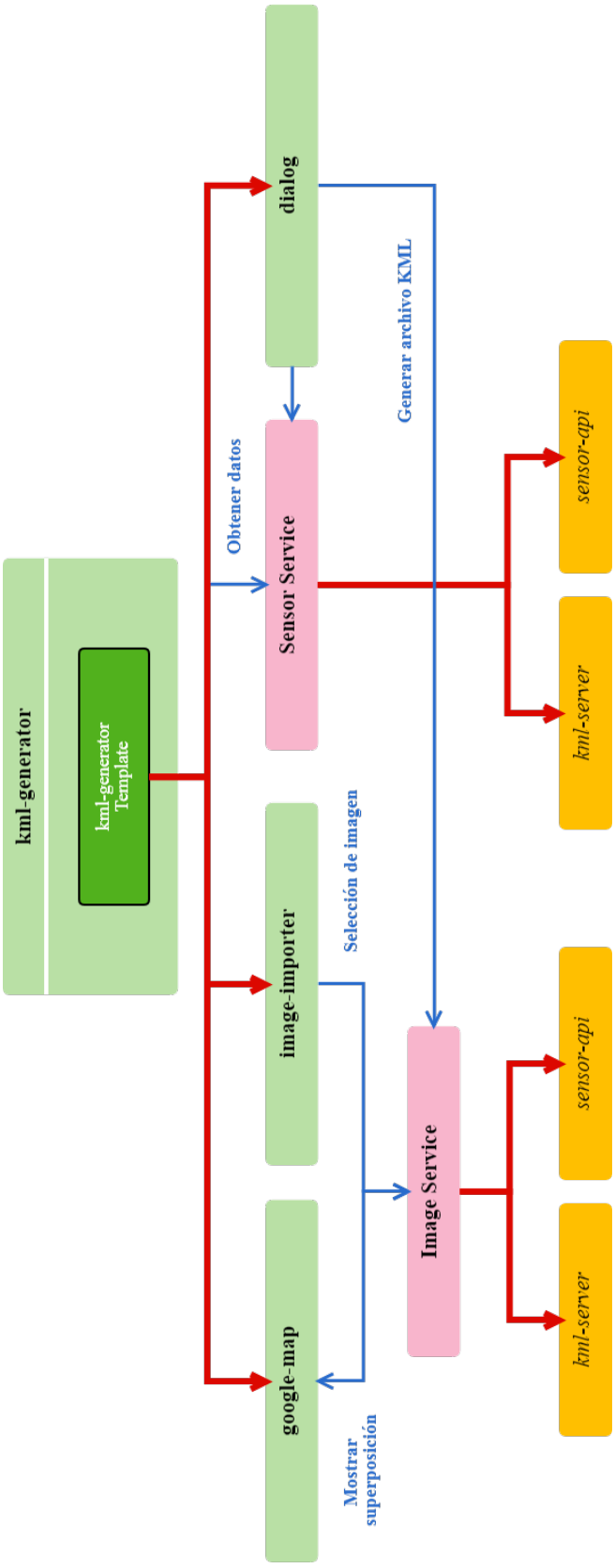


Figura 29: Diagrama del componente kml-generator.

- `sensors-page`: El componente 'sensors-page' está disponible en la ruta '/sensors', y contiene una estructura muy simple dentro de la aplicación, debido a que en su plantilla no hay otros componentes insertados, y solo debe mostrar los datos de los sensores. Éstos datos se obtienen gracias a diferentes métodos del servicio 'Sensor Service'.

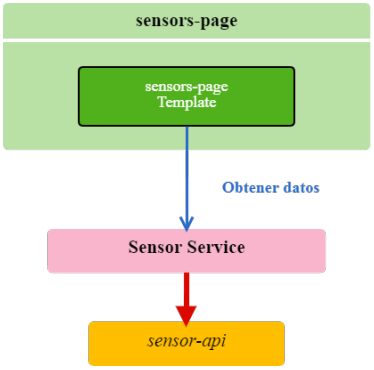


Figura 30: Diagrama del componente sensors-page.

- `gallery-page`: Es el componente que se visualiza al navegar a la ruta '/gallery'. La plantilla de éste componente muestra diversos datos (autores, álbumes e imágenes) que necesitan ser obtenidos mediante algunos métodos del servicio 'Image Service'(que conecta con el servidor 'sensor-api'). Además, 'gallery-page' incluye la funcionalidad de subir una imagen al servidor. Ésta acción es llevada a cabo por el componente 'dialog-image-upload', que también hace uso de los métodos de 'Image Service' para informar al usuario dónde puede subir la imagen (en que autor y en que álbum) y subirla finalmente. Otra funcionalidad de 'gallery-page' es la poder visualizar las imágenes a mayor escala a través del componente 'modal-viewer', que también hace uso de métodos de 'Image Service' para la navegación entre fotografías (poder ir de una fotografía a otra de forma secuencial sin tener que cerrar y abrir el visor 'modal-viewer').

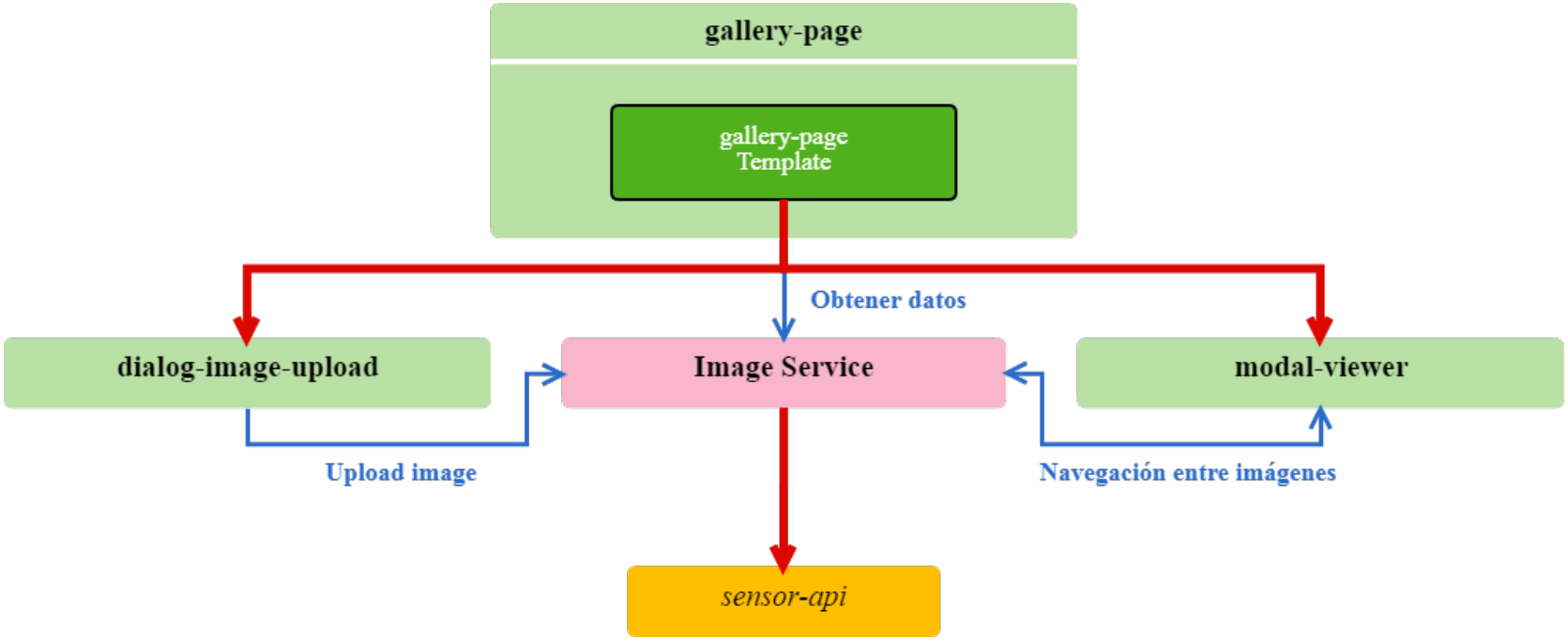


Figura 31: Diagrama del componente gallery-page.

- `demos-page`: El último componente, 'demos-page', se visualiza en la ruta '/demos', y genera su contenido sin necesidad de otros componentes. Su plantilla integra diversos botones con unas funcionalidades que necesitaran llamar a los métodos del servicio 'Demo Service', creado exclusivamente para ésta página.

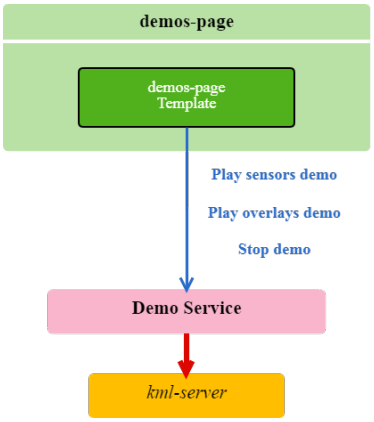


Figura 32: Diagrama del componente demos-page.

En el siguiente apartado se describe el resultado final en la aplicación SAVT-Dashboard a través de las pantallas con las que interactuará el usuario final.

5. Funcionamiento

Todas las pantallas de la aplicación SAVT-Dashboard constan de dos elementos comunes estáticos, es decir, que no varían su contenido y siempre se colocarán en la misma posición:

5.1. Elementos comunes

- Barra de herramientas (figura 33): Situada en la parte superior y con una anchura exacta al de la pantalla, la función de esta barra es la de indicar al usuario el nombre de la aplicación en cualquier momento y la navegación a la página principal cuando se selecciona el nombre, además de tener disponible el botón Menú (a la izquierda del título), que permite abrir y cerrar el menú lateral.

Figura 33: Componente 'Toolbar' de la aplicación 'SAVT-Dashboard'

- Menú lateral (figura 34): Colocado en el extremo izquierdo, engloba toda la altura de la pantalla. Es el componente necesario para la navegación entre páginas de la aplicación. Su contenido se distribuye en una sección superior que muestra el icono del proyecto, una sección a media altura que sirve las cuatro páginas del cliente para que el usuario pueda hacer click en la que desee y navegar hacia ella, y una sección inferior donde se expone el nombre del autor del proyecto y un icono de Github que redirige la página al repositorio abierto Github de la aplicación angular al hacer click en él.

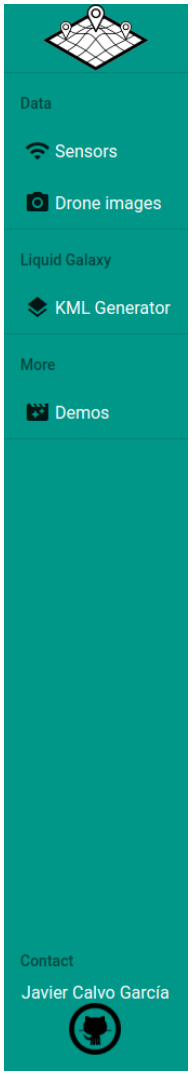


Figura 34: Componente 'Sidenav' de la aplicación 'SAVT-Dashboard'

5.2. Pantalla principal (generador de KML)

La página principal de la aplicación, correspondiente a la figura 35, es la plana en que el usuario puede enviar información a un sistema Liquid Galaxy. Está compuesta por un mapa virtual que alcanza tres cuartas partes de tamaño, ya que es el elemento donde el usuario fijará su atención para poder previsualizar en el mapa virtual los datos que quiera enviar a un Liquid Galaxy.
A la derecha del mapa se encuentran los dos tipos de datos con el que el usuario puede interactuar: en la parte de arriba se sitúan los datos de sensores, y en la parte inferior las fotografías de drones. Cada elemento de estas dos secciones puede ser seleccionado por el usuario, y cuando se produce esta acción se muestra la información elegida en el mapa virtual. La información de las dos secciones se muestra tal cual está organizada su estructura en la base de datos diseñada. En el caso de los datos de sensores, primeramente se muestran todos los campos (`Field` en la base de datos) guardados en el servidor, y una vez el usuario selecciona alguno de estos campos, se presentan todos los sensores (`Sensor`) de aquel campo seleccionado. En el caso de las imágenes, primero se exhiben los autores (`Author`) de las fotografías, después los álbumes (`Album`) del autor seleccionado, y finalmente las imágenes (`Image`) del álbum escogido.
Por último, ésta interfaz consta de un botón centrado en su extremo superior con la función de enviar los datos previamente seleccionados por el usuario. Si se ha seleccionado un dato disponible a enviar, al apretar el botón se abre un diálogo, como el mostrado en la figura 36, que posibilita la opción de confirmar la selección del usuario y enviar finalmente los datos al Liquid Galaxy, o cancelar dicha opción.

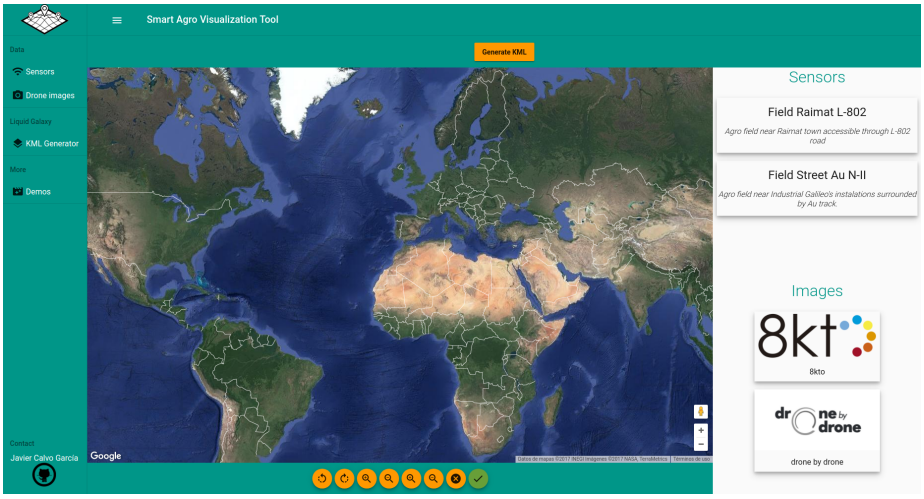


Figura 35: Pantalla principal de la aplicación en la ruta /kml.

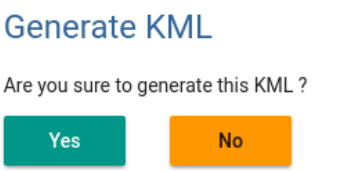


Figura 36: Diálogo de la página principal /kml

5.3. Pantalla sensores

La interfaz la página de sensores (sensors), visible en la figura 37, se compone de dos secciones: selección y visualización. El apartado de selección se sitúa en la mitad superior del plano, dónde se listan todos los campos guardados en la base de datos. El usuario tiene la posibilidad de seleccionar uno de los campos, y la sección converge a un listado de todos los sensores del campo seleccionado. Cada sensor listado puede ser seleccionado para que en el apartado de visualización se muestre toda la información de el elemento elegido.
La sección de visualización, mostrada en la figura 38, es puramente visual, y invisible si un sensor no ha sido seleccionado. En el caso de que se seleccione alguno, muestra toda su información organizada en localización (las coordenadas guardadas del sensor) y valores (valores guardados en la base de datos de las magnitudes medidas por el sensor) de la forma que se observa en la figura 38.

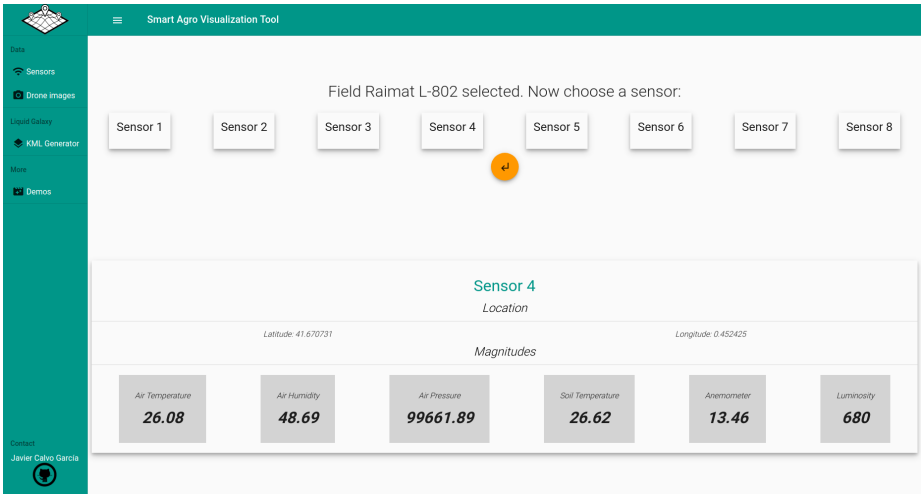


Figura 37: Pantalla de la ruta /sensors

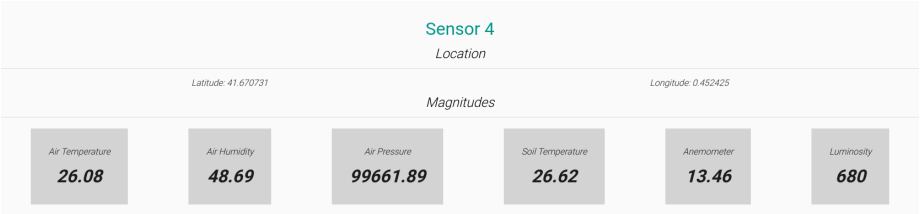


Figura 38: Visualización de los valores de un sensor cuando se selecciona.

5.4. Pantalla galería

La página galería (gallery), como se puede observar en la figura 39, muestra una interfaz de selección similar al de la pantalla de sensores: De forma inicial lista todos los autores (`Author` de la base de datos) disponibles, al seleccionar alguno de éstos se visualizan todos los álbumes de aquel autor, y al seleccionar un álbum, se muestran todas las imágenes (`Image`) de el álbum (`Album`) elegido. Al hacer click sobre alguna imagen, se despliega en primer plano de toda la pantalla un visor de imágenes para poder ver las fotografías con mas detalle. Además, para una mejor experiencia de usuario, el visor permite navegar de forma secuencial entre las imágenes sin necesidad de ser cerrado.
Además, la galería ofrece la posibilidad de subida de imágenes mediante el botón circular situado en la parte superior de la página, el cual abre una ventana, como en la figura 41, que permite subir un imagen del sistema y destinarla a su autor y álbum correspondiente.

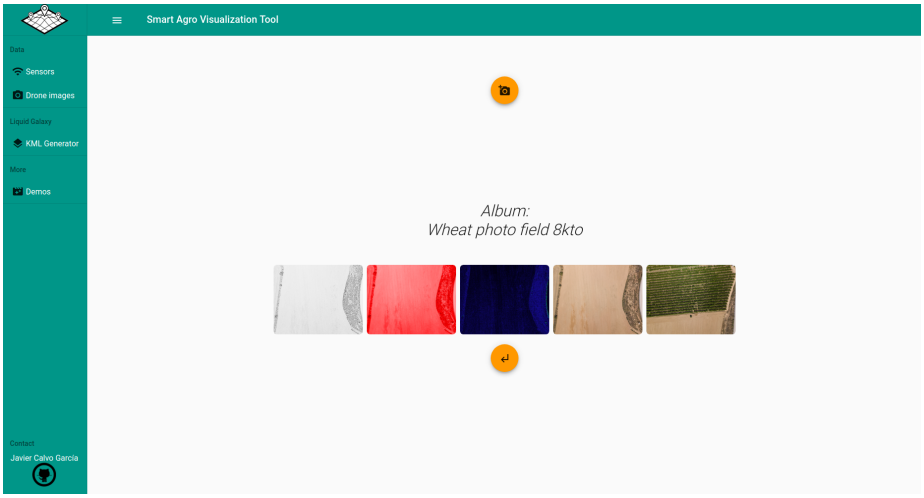


Figura 39: Pantalla de la ruta /gallery

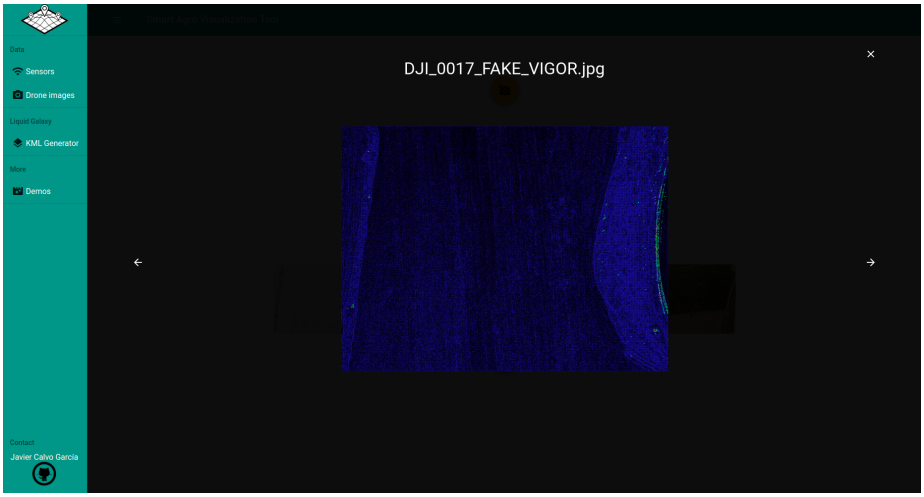


Figura 40: Visor de imágenes.

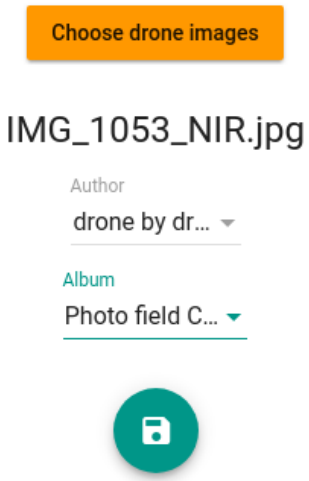


Figura 41: Diálogo de subida de una imagen en la página Galería

5.5. Pantalla demostraciones

La última pantalla de la aplicación (figura 42) muestra tres botones junto a su descripción distribuidos en el centro de la página que, al seleccionarlos ejecutaran las llamadas pertinentes para generar las demostraciones (demos) en el Liquid Galaxy.
En orden vertical, los dos primeros botones crearan una demostración de datos de sensores y fotografías (en forma de recorrido llamado *tour*') respectivamente, mientras que el tercer botón ejecuta el comando para parar y eliminar dicha demostración.

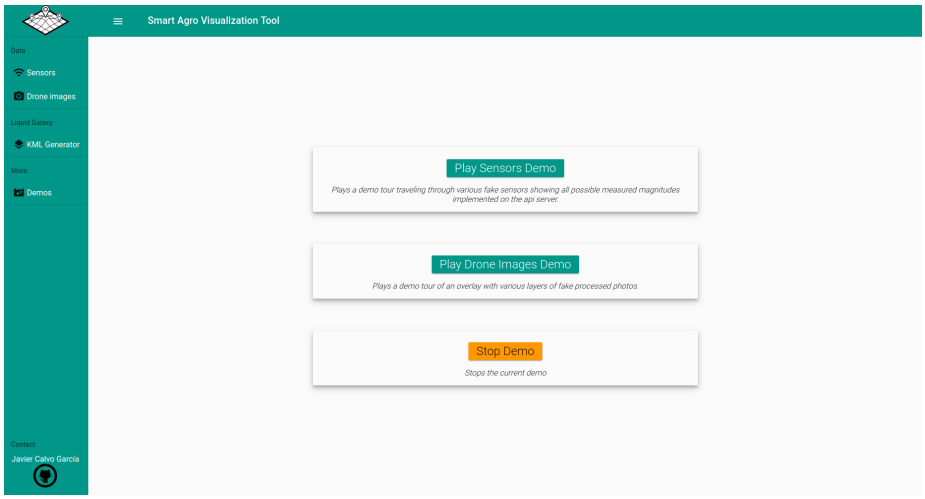


Figura 42: Pantalla de la ruta /demos

Como último apartado de este informe, el autor de este proyecto habla en primera persona sobre las conclusiones personales del mismo, y describe las posibles líneas de futuro que podría tomar el desarrollo para la inserción de mejoras y nuevas características.

6. Conclusión

Gracias al desarrollo de este proyecto he conseguido aprender (y en cierto grado dominar) los entornos de trabajo NodeJS y Angular. Estas tecnologías ya habían sido vistas antes de desarrollar Smart Agro Visualization Tool en el grado de Ingeniería Informática, y gracias a la producción de las aplicaciones del proyecto, se han profundizado los conocimientos sobre ellos.

Respeto al informe del proyecto, ha sido una parte costosa de concebir, ya que la escritura (específicamente la descripciones de las estructuras de las aplicaciones y su funcionamiento) es uno de mis puntos débiles, además que no es algo que disfrute haciendo. De todas formas, se ha intentado redactar un documento de forma simple que pueda ser leído por usuarios expertos y no expertos en el contexto, indicando siempre la definición de cada concepto técnico, así como la aportación de un grupo de diagramas y gráficos de elaboración propia que ayuden al lector a crear un esqueleto gráfico de los elementos que se describen.

Personalmente me siento satisfecho del trabajo realizado. Hasta ahora no había realizado ningún proyecto desde cero de estas dimensiones, pero los conocimientos previos estudiados en el grado me han servido para poder cumplir con los objetivos básicos. Además, he podido implementar las aplicaciones de forma que puedan ser desplegadas en cualquier servidor que cumpla con los requisitos, y todas ellas son escalables en un futuro con nuevas funcionalidades. A continuación se listan las líneas abiertas de futuro que pueden tener las aplicaciones de el proyecto Smart Agro Visualization Tool.

6.1. Líneas abiertas

Una aplicación web puede tener una gran cantidad de lineas futuras que abarcar, ya que los datos con los que interactúa están en continuo cambio, además de estándares de navegación y herramientas de navegadores que evolucionan con el paso del tiempo. Para éste proyecto, se han ideado diversas funcionalidades que pueden ser de gran utilidad para los usuarios de sus aplicaciones, son las siguientes:

- Añadir componentes estadísticos para los valores de sensores: Una posible vía futura de trabajo sería la de incorporar en la página del sensor (o crear una nueva) elementos gráficos de estadística de los valores. Ésta funcionalidad se ideó y intentó implementar durante el desarrollo del proyecto, pero las librerías utilizadas dieron diversos problemas y se decidió dejar su programación como trabajo futuro.
- Mejorar interacción mapa-imágenes de la página principal: Otra funcionalidad que puede ser desarrollada en el futuro es la de modificar la manera que tiene el usuario de añadir los datos al mapa virtual del cliente (en la página principal de la ruta /kml), con la inserción de componentes diferentes dentro de la interfaz que se simplifiquen ésta acción.

- Otorgar herramientas de administración al usuario: Otra de las características que se podría añadir a la aplicación es la de proporcionar al usuario de la aplicación cliente algún mecanismo para que tenga control sobre las interacciones con el servidor, como pueden ser que se pueda determinar las direcciones ip a través de la interfaz y no a partir de los comandos del sistema, o que pueda eliminar, modificar o incluso crear información de las base de datos desde la misma interfaz.

- Mejorar eficiencia de carga de imágenes: Durante el desarrollo del proyecto se pudo comprobar que si se desplegaba el servidor de datos en un servidor público, la carga de las fotografías en el cliente y su envío al Liquid Galaxy era lenta, por lo que una linea de trabajo futura puede ser la mejora de este aspecto. Esto podría realizar-se mediante cambios en el almacenaje de imágenes, con la subida de diferentes resoluciones o cargas de miniaturas en casos que la medida original de la fotografía no sea necesaria.

Bibliografía

[1] Dacom. Dacom farm products - terrasen. https://dacom.farm/products/terrasen.

[2] Blog LleidaDrone. Aplicaciones de los drones en la viticultura. http://www.lleidadrone.com/2015/04/aplicaciones-de-los-drone-en-la.html.

[3] Hemav. Agricultura de precisión. https://hemav.com/servicio/agricultura-de-precision/.

[4] Google Earth. Escaparate galaxia líquida. https://www.google.com/earth/explore/showcase/liquidgalaxy.html.

[5] NEC. Nec and dacom collaborate on precision farming solution. http://www.nec.com/en/press/201410/global_20141023_03.html.

[6] Blog Internet of Things Agenda. What is the internet of things (iot) ? http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT.

[7] Blog Hipertextual. ¿qué es y cómo funciona el internet de las cosas? https://hipertextual.com/archivo/2014/10/internet-cosas/.

[8] FIWARE. About us (sobre nosotros). https://www.fiware.org/about-us/.

[9] Liquid Galaxy. Github wiki liquid galaxy. https://github.com/LiquidGalaxy/liquid-galaxy/wiki.

[10] Libelium. Página oficial. http://www.libelium.com/.

[11] Waspmote. Documentación oficial del producto. http://www.libelium.com/development/waspmote/documentation/.

[12] Wikipedia. Definición plano cenital. https://es.wikipedia.org/wiki/Plano_cenital.

[13] Wikipedia. Definición liquid galaxy. https://en.wikipedia.org/wiki/Liquid_Galaxy.

[14] Wikipedia. Definición google earth. https://en.wikipedia.org/wiki/Google_Earth.

[15] MongoDB. Introducción a mongodb. https://docs.mongodb.com/getting-started/shell/introduction/.

[16] MongoDB. Guía de introducción. https://docs.mongodb.com/getting-started/shell/introduction/.

[17] Wikipedia. Definición node.js. https://es.wikipedia.org/wiki/Node.js.

[18] Node.js. About. https://nodejs.org/es/about/.

[19] ExpressJS. Página principal. http://expressjs.com/.

[20] mongoose. Página principal. http://mongoosejs.com/.

[21] Wikipedia. Definición typescript. https://en.wikipedia.org/wiki/TypeScript.

[22] Wikipedia. Definición angular. https://es.wikipedia.org/wiki/Angular_(framework).

[23] Angular. Guía oficial. descripción ngmodule. https://angular.io/guide/ngmodule.

[24] Angular. Guía oficial. descripción sintaxis de plantilla. https://angular.io/guide/template-syntax.

[25] Wikipedia. Definición dom. https://en.wikipedia.org/wiki/Document_Object_Model.

[26] Wikipedia. Definición python. https://es.wikipedia.org/wiki/Python.

[27] Wikipedia. Definición gnu lesser general public license. https://es.wikipedia.org/wiki/GNU_Lesser_General_Public_License.

[28] Github. Repositorio angular cli. https://github.com/angular/angular-cli/wiki.

[29] Angular. Glosario. definición jit. https://angular.io/guide/glossary#jit.

[30] Wikipedia. Definición exif. https://en.wikipedia.org/wiki/Exif.

[31] Github. Repositorio proyecto multer. https://github.com/expressjs/multer.